ARI Research Note 86-26

AD-A168 635

ADVANCED TERRAIN REPRESENTATION FOR THE MICROTICCIT WORKSTATION: SYSTEM MAINTENANCE MANUAL

Decisions and Designs, Inc.

for

Contracting Officer's Representative
Donald M. Kristiansen

ARI Field Unit at Fort Knox, Kentucky
Donald F. Haggard, Chief

TRAINING RESEARCH LABORATORY
Seward Smith, Acting Director

U. S. Army

Research Institute for the Behavioral and Social Sciences

February 1986

U. S. ARMY RESEARCH INSTITUTE
FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency under the Jurisdiction of the

Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

WM. DARRYL HENDERSON
COL, IN
Commanding

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARI Research Note 86-26	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ADVANCED TERRAIN REPRESENTATION FOR THE MICRO-TICCIT WORKSTATION: SYSTEM MAINTENANCE MANUAL		5. TYPE OF REPORT & PERIOD COVERED Interim Report September 84 - May 85
		6. PERFORMING ORG. REPORT NUMBER DDI/MM 85-17-357
7. AUTHOR(s) Decisions and Designs, Inc.		8. CONTRACT OR GRANT NUMBER(s) MDA 903-84-C-0333
9. PERFORMING ORGANIZATION NAME AND ADDRESS Decisions and Designs, Inc., Suite 600, 8400 Westpark Drive, P.O. Box 907 McLean, Virginia 22101		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2Q263743A794
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Institute for the Behavioral and Social Sciences, 5001 Eisenhower Avenue, Alexandria, VA 22333-5600		12. REPORT DATE February 1986 13. NUMBER OF PAGES 251
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) - -		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE - -

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

- -

18. SUPPLEMENTARY NOTES

Donald M. Kristiansen, contracting officer's representative and technical monitor.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Land Navigation
Surrogate Travel
Computer-Assisted Instruction
Armor Training

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report provides a technical description of the MicroTICCIT version of Decision and Design, Inc.'s (DDI) Advanced Terrain Representation (ATR) system. Topics covered in the report include theory of operation, installation, basic operations, and troubleshooting. The MicroTICCIT version of ATR is a device for training land navigation. This training is accomplished by having students use the system to "travel" over simulated terrain, review prerequisite skills, receive lessons in dead reckoning, and perform practice and test problems (over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ARI Research Note 86-26

20. Abstract (continued)

in navigation from one point on the ground to another.

ATR is a microcomputer-based technology that utilizes laser videodiscs to provide a capability for fully interactive surrogate travel.

Page E-8 in this report should be deleted.
Per Mr. Douglas Edwards, Army Res. Inst. for the Behavioral and Social Sciences

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

FIGURES

TABLES

1.0 INTRODUCTION

The purpose of this manual is to provide a technical description of the MicroTICCIT version of Decisions and Designs, Inc.'s (DDI) Advanced Terrain Representation (ATR) System. Topics covered in this manual include theory of operation, installation, basic operation and troubleshooting. A less technically oriented version of system functions and operating instructions may be found in MicroTICCIT Courseware for "Navigate from A to B": Instructor's Guide, published concurrently with this Maintenance Manual.

1.1 System Functions

The MicroTICCIT version of ATR is a device for training land navigation. The system documented here trains for the task "Navigate from A to B". This training is accomplished by having students use the system to "travel" over simulated terrain, review prerequisite skills, receive lessons in terrain association and dead reckoning, and perform practice and test problems in navigation from one point on the ground to another.

1.2 System Overview

ATR is a microcomputer-based technology that utilizes laser videodiscs to provide a capability for fully interactive surrogate travel. The version of ATR documented here is implemented on a MicroTICCIT workstation for use in land navigation training. Certain additional components are required to enable the basic MicroTICCIT workstation to support ATR.

Running MicroTICCIT ATR requires a system with seven hardware components, as seen in Figure 1-1. Four of these components form the MicroTICCIT workstation:

1. The MicroTICCIT terminal, an IBM PC modified by Hazeltine. To run ATR, the terminal must be equipped with at least one floppy disk drive and 128K Bytes of memory.

2. The MicroTICCIT keyboard.

3. A Sony PVM1270Q monitor.

4. A Sony LDP-1000 videodisc player (this is optional for a MicroTICCIT workstation, but essential for ATR).

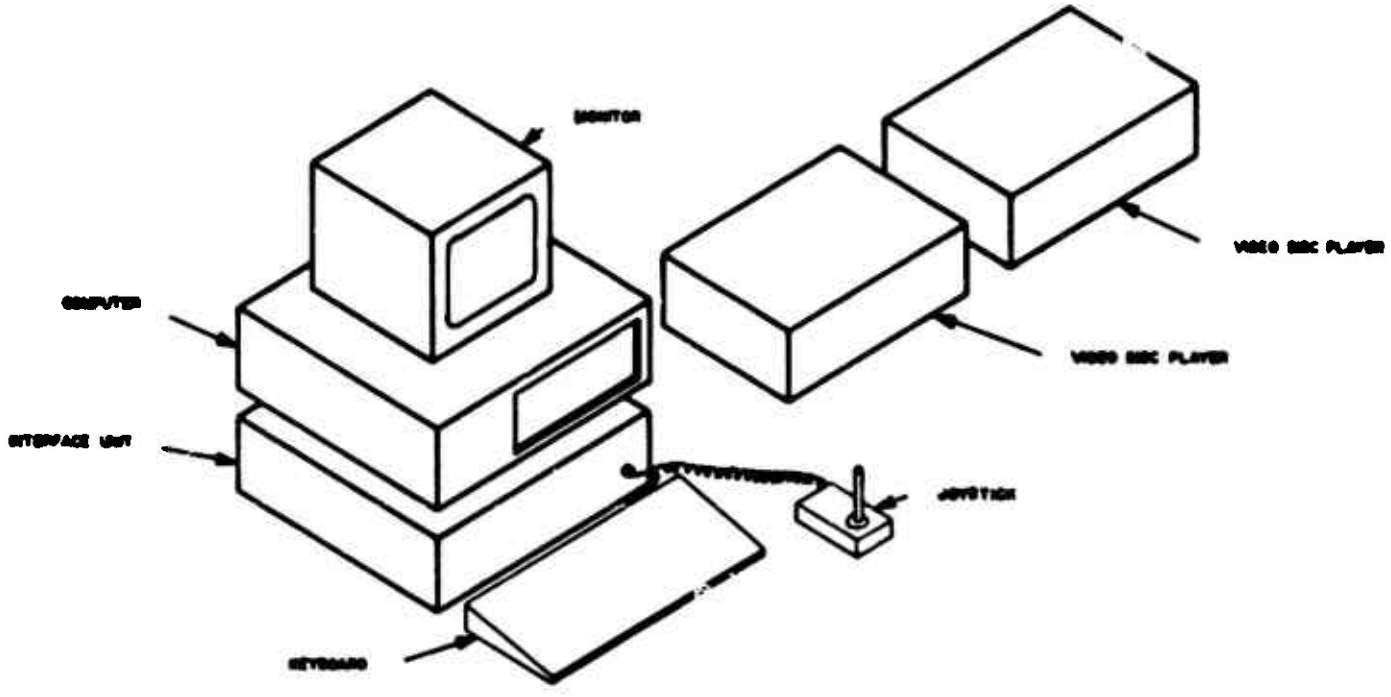


Figure 1-1. ATR SYSTEM

The last three additional components are:

5. A second Sony LDP-1000 videodisc player.

6. A Wico 50-0299 joystick/keypad.

7. An Interface Unit, designed and built by DDI, to provide communications among the computer, the videodisc players, and the joystick.

1.3 Performance Specifications

The ATR additions to the MicroTICCIT workstation are designed for operation in a normal classroom environment. Table 1-1 gives limits for the operating parameters.

PARAMETER	LIMITS
Power	110-120 V AC, 60 Hz
Temperature	65° - 85° F
Humidity	30%-90%, non condensing
Static Protection	Not Needed
Vibration	± 0.25G

Table 1-1. LIMITS FOR OPERATING PARAMETERS

2.0 HARDWARE DESCRIPTION

The purpose of this chapter is to describe the hardware which makes up the MicroTICCIT version of the ATR System and to provide a theory of operation for the system and its sub-assemblies.

2.1 System Description

The ATR System is designed to lead a student through an interactive training session. The hardware of the ATR System supports this by presenting the student with text and video imagery and allowing the student to interact with the system by joystick and keyboard commands. A novel feature of the ATR System is the use of two videodisc players containing identical disks to provide smooth transitions between video frames. The DDI Interface Unit coordinates the videodisc players to provide this function. Additionally, the Interface Unit controls the configuration of the system.

There are two configurations for the ATR System. When the Interface Unit is turned on, the system is as shown in Figure 2-1. In this configuration the MicroTICCIT terminal can run the ATR software, select video from either videodisc player and receive input from both the keyboard and the joystick. When the Interface Unit is turned off the system reverts to the original workstation configuration (Figure 2-2). In this configuration the MicroTICCIT terminal can receive video only from videodisc player 1 and the joystick cannot be used. The ATR software will not run properly when the Interface Unit is turned off. The reconfiguring is accomplished by a bypass module within the Interface Unit and does not require any cable changes. Unless otherwise specified the information in this manual applies to the full configuration.

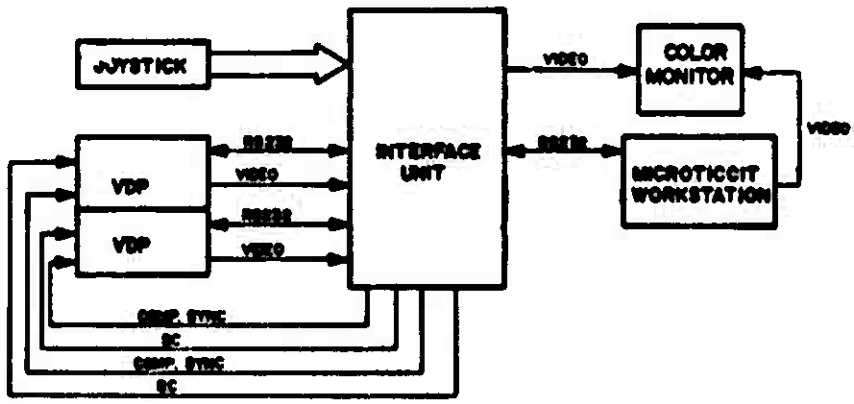


Figure 2-1. FULL CONFIGURATION

Table 2-1 also shows the types and paths of signals between the subassemblies. Each of the subassemblies will be described below.

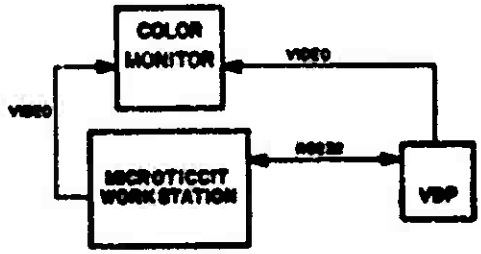


Figure 2-2. WORKSTATION CONFIGURATION

2.2 Subassembly Descriptions

The subassemblies which make up the ATR System are made by a variety of manufacturers including DDI, IBM, Sony, Pioneer, and Wico. All of the subassemblies will be described briefly; only the DDI Interface Unit will be described in detail. For further information on other products refer to the appropriate reference manual.

2.2.1 MicroTICCIT terminal - The MicroTICCIT terminal is the controller for the ATR System. It is basically an IBM PC which has been modified by the addition of a custom color graphics board. To support ATR the terminal must have 128K of memory and at least one floppy disk drive. The terminal communicates with the rest of the ATR system through a single RS-232C serial port and a video output channel. Additional information on the MicroTICCIT terminal can be found in the IBM Personal Computer Technical Reference Manual.

2.2.2 Sony monitor - The display for the ATR System is a Sony Model PVM-1270Q color monitor. The monitor has 525 line x 512 pixel resolution and accepts a NTSC composite video signal.

2.2.3 Laser videodisc players - The terrain video images are read from videodisc by a pair of Sony LDP-1000A videodisc players. These are top loading units and should not have anything stacked on them. Both players contain identical discs and are controlled so that one player is displaying a frame of video while the other player is seeking the next frame of video to be displayed.

Some ATR Systems will be equipped with Pioneer LD-V6000 videodisc players. These front loading players can be stacked to reduce the space requirements for the ATR System. All other differences are transparent to the user.

2.2.4 ATR Interface Unit - The Interface Unit ties the various components of the system together. It multiplexes the RS-232 data lines, selects the video signal to be displayed, generates composite sync and subcarrier signals for the videodisc players and converts the joystick signals tc RS-232 format. Figure 2-3 is a detailed block diagram of the ATR system which shows the internal components of the Interface Unit and the signals between them and the rest of the ATR System.

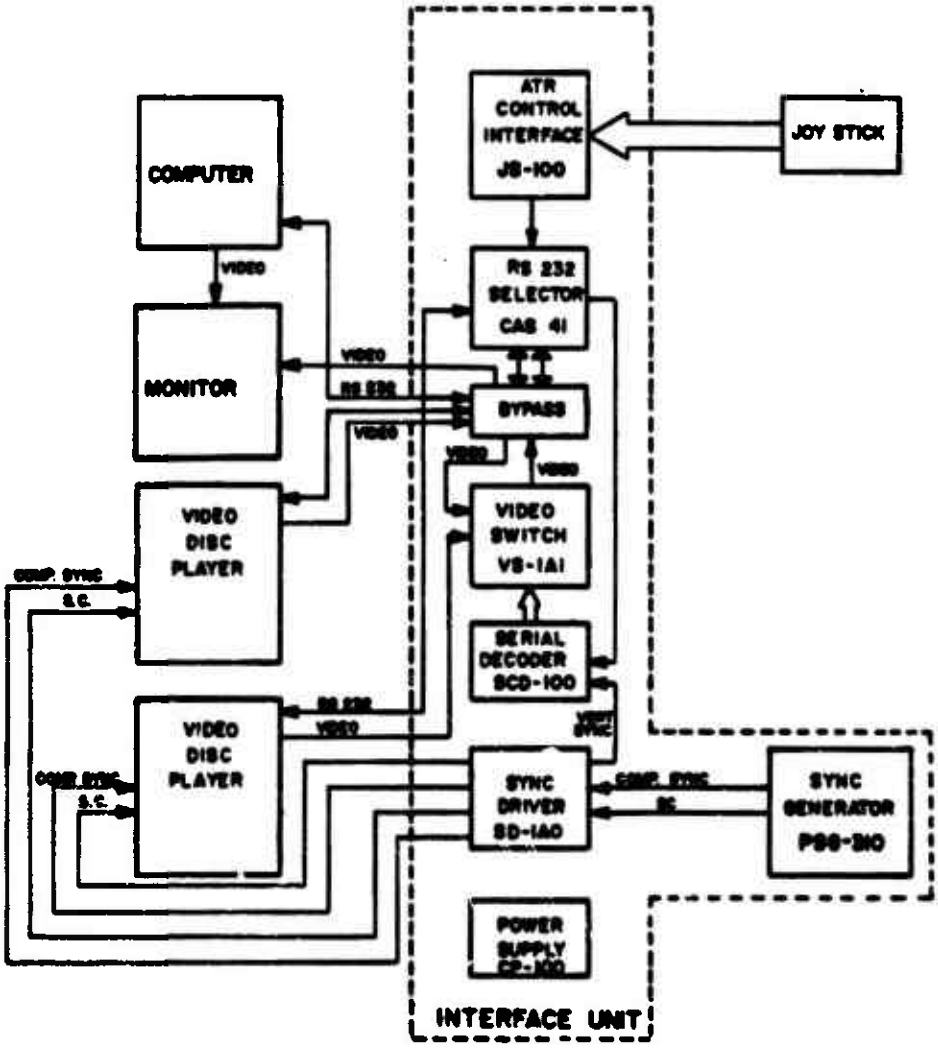


Figure 2-3. ATR INTERFACE UNIT BLOCK DIAGRAM

Figure 2-4 shows the physical layout of the Interface Unit. There are three major sections, the logic rack containing six circuit cards plugged into a motherboard, the CAS-41 RS-232 selector and the relays, K1 through K3, which make up the Bypass Module.

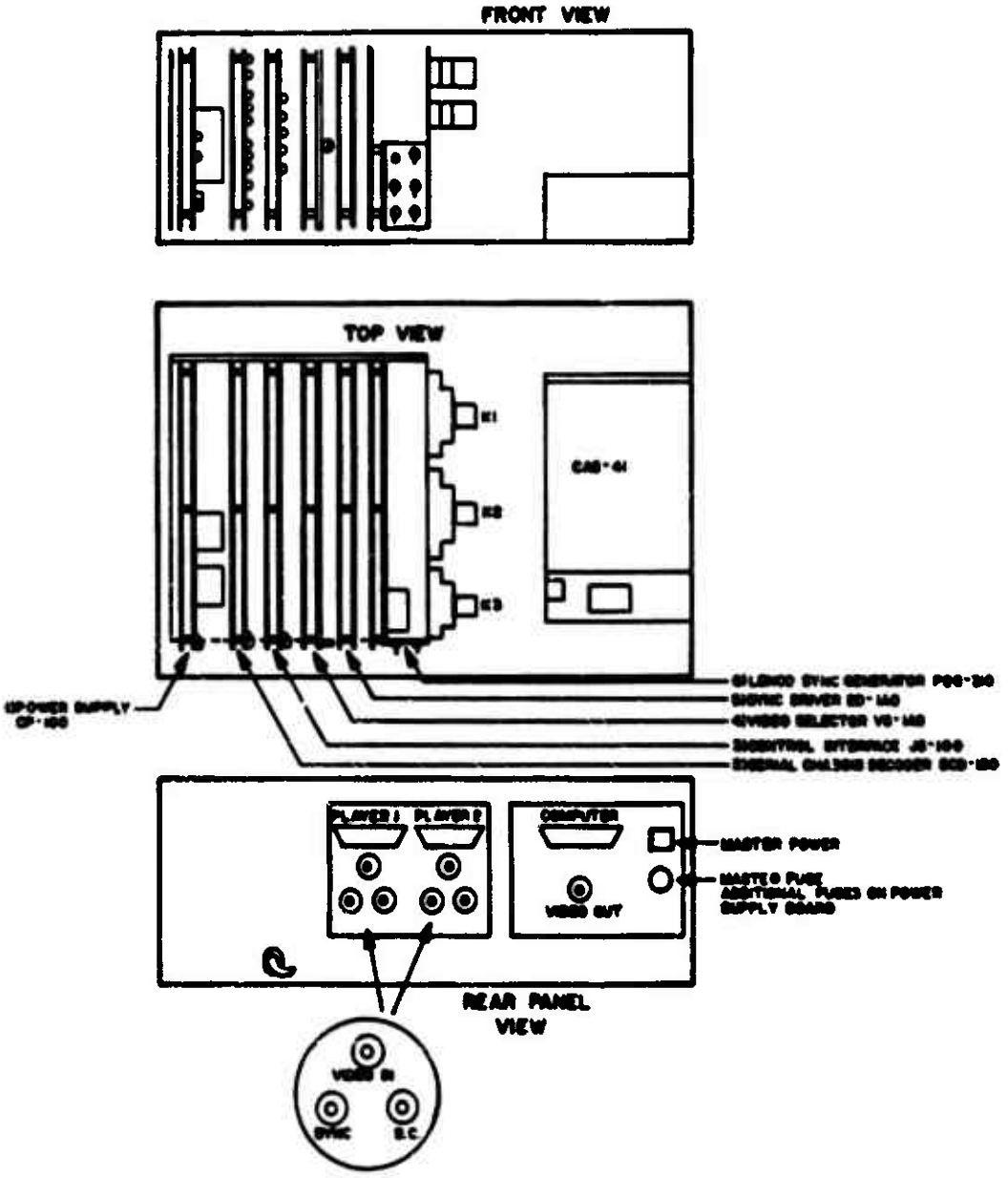


Figure 2-4. ATR INTERFACE UNIT PHYSICAL LAYOUT

2.2.4.1 CP-100 Power Supply - This card provides DC power for the logic rack. Input is 110V AC, outputs are +18V, -18V and GND. The outputs are not regulated. There are fuses on the ACV, +V and -V lines. The LEDs on the front of the power supply indicate that the power supply is on and that the fuses are intact (Figure 2-5).

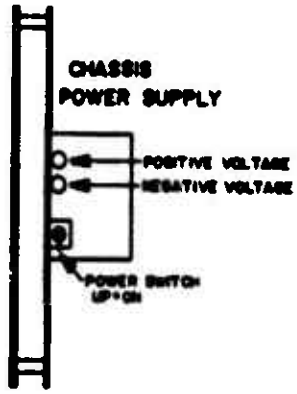


Figure 2-5. CP-100 POWER SUPPLY

2.2.4.2 SCD-100 Serial Chassis Decoder - This card receives RS-232 commands, interprets them and sends control signals down the motherboard to video selector cards. In the ATR system these commands select which videodisc player signal to display. The SCD-100 uses a vertical sync input for timing so that the video switching takes place between frames of video. This prevents the picture tear which would occur if the video were switched in the middle of a frame. The LEDs on the front provide a readout of the stop bit, start bit and the eight bits which make up a command word (Figure 2-6).

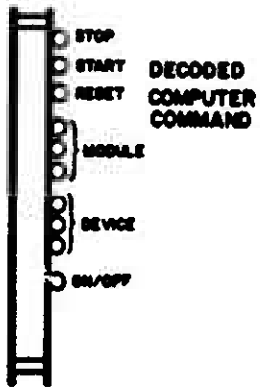


Figure 2-6. SCD-100 SERIAL CHASSIS DECODER

2.2.4.3 JS-100 Control Interface - The JS-100 is the interface between the joystick and the rest of the ATR System. The circuit card takes the six bits of parallel data produced by the joystick and transmits them as RS232 serial data at 1200 baud. The six bits of joystick data are padded to eight bits so that the joystick command can be interpreted as a legitimate ASCII character by the MicroTICCIT terminal. The LEDs on the front of the card indicate the values for the six parallel joystick lines (Figure 2-7).

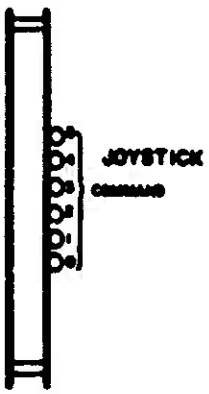


Figure 2-7. JS-100 CONTROL INTERFACE

The Wico joystick combines both directional joystick and numeric keypad functions. All outputs from the joystick are digital; the output is not proportional to the amount of stick movement.

2.2.4.4 VS-1A8 Video Selector - This circuit card selects, buffers and outputs one of up to eight input video signals. The selection is controlled by a command sent down the bus by the Serial Chassis Decoder described above. There is a potentiometer on the front of the card which adjusts the gain of the output amplifier (Figure 2-8). This allows the output to be set to 1 Volt peak-peak as specified by the NTSC standard. Note that in the ATR System only two of the input lines are used.

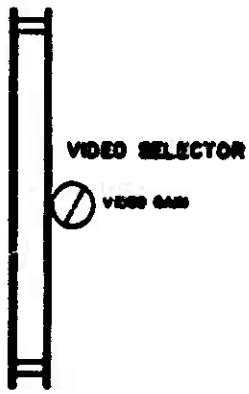


Figure 2-8. VS-1A8 VIDEO SELECTOR

2.2.4.5 SD-1A0 Sync Driver - This card receives composite sync and subcarrier signals as input. It produces four buffered composite sync signals, one vertical sync signal generated from composite sync and two buffered subcarrier signals as outputs.

2.2.4.6 PSG-310 Sync Generator - The Sync Generator card is the master video timing source for the ATR System. The PSG-310 produces composite sync and subcarrier signals which lock the videodisc players together in time. This is necessary for smooth video switching. There are several controls on the front

of the card, all of which should be adjusted only by qualified personnel (Figure 2-9). Refer to the Lenco PSG-310 Instruction Manual for a detailed explanation of the controls.

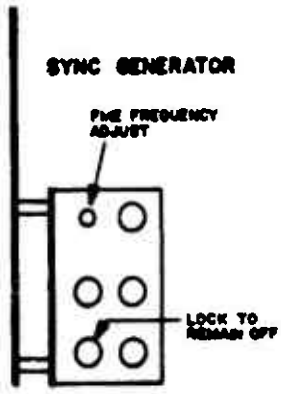


Figure 2-9. PSG-310 SYNC GENERATOR

2.2.4.7 CAS-41 Serial Selector - The CAS-41 is a microcomputer controlled RS 232 selector built by Western Telematics Inc. The unit was designed to permit one serial device to select and communicate with one of four serial devices. As it is used in the ATR System the CAS-41 allows the MicroTICCIT terminal to talk to either the serial decoder card, the control interface card or one of the videodisc players. Figure 2-10 shows the CAS-41 and the internal serial cables for the ATR Interface Unit. Note that there is bidirectional communication only between the computer and the videodisc players; the other cables are one directional.

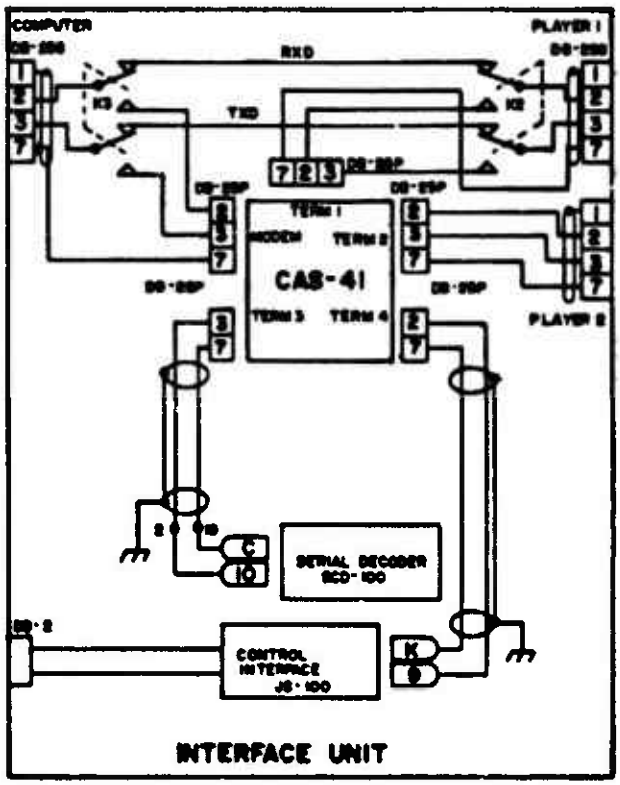


Figure 2-10. CAS-41 SERIAL SELECTOR AND CABLES

2.2.4.8 Bypass module - The bypass module switches the ATR System between the full configuration and the MicroTICCIT workstation configuration. It consists of three DPDT relays, two for serial switching and one for video. These are 12 Volt relays which draw their power from the CP-100 power supply. They are shown as K1, K2 and K3 on Figure 2-4.

3.0 SOFTWARE AND VIDEODISC DESCRIPTION

This chapter describes the software and videodiscs used to provide surrogate travel and training in the MicroTICCIT version of ATR.

3.1 Operating System

The system runs under IBM PC DOS, Version 2.1, with additional device drivers for the MicroTICCIT keyboard and monitor. The operating system and the MicroTICCIT modifications are installed on the floppy diskette that contains the system software/courseware, and are "transparent" to the user.

3.2 ATR Videodisc

Each copy of the ATR videodisc contains nearly 53,000 photographs of simulated terrain which are presented on the monitor during surrogate travel. The simulated terrain was photographed in an orderly fashion using a grid layout; storage of the photographs on the videodisc follows the photography pattern. Photographs were taken at intervals equivalent to twelve meters on real terrain. An area equivalent to 648 by 732 meters was covered, with sixteen photographs (one each facing north, north-northwest, northwest, west-northwest, west, etc.) taken at each point or grid center. Since there are 54 twelve-meter intervals in 648 meters, and 61 twelve-meter intervals in 732 meters, 54x61x16, or 52,704 photographs were taken. The sixteen photographs at every point allow the user to look in any of sixteen directions from any point on the terrain, and to simulate pivoting through a full 360 degrees. Movement through the terrain can be simulated by presenting appropriate photographs from nearby grid centers.

3.3 Program Description

The basic ATR software enables the hardware to access and display video images in such a fashion that travel is simulated. In the MicroTICCIT version, surrogate travel is augmented with courseware to train land navigation. These two functions are discussed in the following sections. A commented source listing appears in Appendix B.

3.3.1 Software for surrogate travel - Simply put, the surrogate travel software translates instructions from the user interface (joystick) into instructions for the videodisc players in order to simulate travel over the ATR terrain. The orderly layout of the photographs on the videodisc permits an equally orderly software representation for the images, and this in turn makes the translation of user instructions to hardware instructions relatively straightforward.

3.3.1.1 Software representation of terrain - The grid and directions in the simulated terrain correspond to a table in the ATR software. There are 54 east-west positions, and 61 north-south positions, for a total of 3294 grid centers. Sixteen directions are represented for each grid center. Within the software, a border or buffer of two positions was added around the parameter of the simulated terrain. Hence there are 58 possible values for the x, or east-west, coordinate and 65 possible values for the y (north-south) coordinate. A third coordinate, z, represents direction, and can take on any of sixteen possible values. The table contains one bit for each possible combination of x, y, and z values.

One purpose of the table is to determine the legality of any position to which the user might try to travel. All of the buffer positions are illegal, since they are entirely theoretical. Certain other positions are made illegal within the software because, for various reasons (e.g.,

inability to take a photograph because of physical obstructions), no photographs are available for these positions. Each illegal position has a value of "1" in the table, while legal positions are assigned a value of "0".

Coordinates in the table serve as inputs to computation of the videodisc frame number that must be accessed to simulate movement to a legal position.

3.3.1.2 Surrogate travel functions - At any time that the ATR surrogate travel software is in use, there is exactly one current value for XYZ, corresponding to one photograph on the ATR videodisc. X and Y values are translated into six-digit coordinates, a specific location on the terrain map, and Z values are translated into compass bearings, the direction of view. Given joystick communication from a user, the ATR software decodes the instruction and uses it to compute a provisional new XYZ value, six-digit coordinate, and compass bearing. If the new XYZ value corresponds to a legal position (one having a value of "0" in the frame table), the software updates XYZ and sends out an instruction for a videodisc player to access the appropriate photograph. Each time the X and Y values change, the distance traveled from the old XYZ to the new XYZ is added to an odometer reading. The compass, odometer, and coordinates appear on a status line at the bottom of the screen. (Compass bearings are given as magnetic azimuths and odometer readings are given in meters.) If the position is illegal, the software instructs the hardware to beep as a signal that the user's instruction cannot be carried out, and XYZ retains its prior value.

In addition to translating instructions from the user to videodisc frame numbers, the surrogate travel software determines which videodisc player should present a given photograph, and when the photograph should be presented. Because identical videodiscs are played on both players, either player can access any legal video frame. However, since a

videodisc player that is searching for an image cannot display an image, it is desirable to alternate players, so that there is always a picture present on the monitor. For the same reason, it is also desirable to wait until the next image has been found before switching away from the current one. The software tracks the activities of the two players and monitors the status of the searching player; this information allows the software to instruct the Interface Unit to display the appropriate video source (player) after the new frame has been found.

Another timing function served by the surrogate travel software is the insertion of delays. The time required for a videodisc player to find a new frame depends on the "distance" or number of video frames on the disc between the current frame and the new frame. Because of the layout of photographs on the videodisc, the smallest distance between current and new frames occurs when the user is moving due north or due south. Travel in other directions involves moving across more interposed frames of video (the exact number depends on travel direction). To correct the resulting inconsistency in access time, the software inserts a brief delay before instructing the interface module to switch to a new frame that is accessed more quickly than others.

3.3.2 Courseware For "Navigate from A To B" - The courseware consists of four major components. Each component is described below.

Free Travel - This function allows the user to travel freely over the terrain by using a joystick. The system uses input from the joystick to calculate coordinates and frame numbers for the videodisc player.

Skills Review - In this component, text is displayed which reviews the skills that the student must understand completely before proceeding to the land navigation training section.

The courseware text prcmpts the user for specific information. The system reads the information, compares the user's input with the correct answer to the question posed and provides the proper response to the user. The system also contains erroneous responses which the user could make due to careless mistakes. If the user gives one of these "smart" answers, the system responds with a statement which may help the user correct a mistake.

Navigate From A to B - This component leads the student through the 19K BNCOC Lesson Plan (Task No. 071-329-1006), by presenting text which explains two methods of land navigation, terrain association and dead reckoning. The system accesses and presents a previously planned route of travel to demonstrate each method of land navigation. Within each method, a second route of travel is described to the user, but the user is allowed to travel the route as in free travel. The system can determine the location at which the user stopped and provide feedback to the user according to his performance in navigation.

Navigation Test - This section contains the test for "Navigate from A to B". A password and student information must be entered before the system allows access to the actual test administration. The test contains three problems in which the system provides the coordinates of the start point and release point for each problem. The user is permitted to travel freely to navigate a route to the destination. After deciding that he has found the destination, the user presses the button on the joystick. The system determines the location at which the user stopped to evaluate the user's ability to navigate from one point on the ground to another. To PASS a test problem, the user must be able to navigate from point A to within 50 meters of point B; otherwise the user receives a FAIL on that navigation problem. The user is given a GO after having passed

two of the three test problems on the test. If the user does not receive a pass on two problems, a NO GO is given on that test. The user may take the test again, but the test may only be given to the same user a total of three times. The system displays the results of each problem and determines whether a GO or NO GO should be entered into the student's Master Record. Another password must be entered to free the system after the test has been taken.

Shut Down ATR System - This provides an explanation of the switch settings on the videodisc players. The switches must be set as indicated for TICCIT to operate properly. Failure to reset the switches as indicated will degrade the performance of the MicroTICCIT workstation.

4.0 INSTALLATION

The ATR System can be set up on any flat stable surface, requiring approximately the same space as the MicroTICCIT workstation. The system can be asaembled by one peraon without special tools. A small screwdriver may be useful.

4.1 Unpacking

The ATR System will arrive in three boxes, one each for the Interface Unit and videodisc player and one box for cablea, accessoriea and manuals. Unpack all of the boxes, being careful to remove all of the cables from the packing materials. Table 4-1 givea a list of all of the components needed for an ATR System. It is advisable to save the videodisc player and Interface Unit boxes for returning subasaemblies to DDI for possible future repair.

PART	QUANTITY
Hazeltine MicroTICCIT Terminal	1 Customer Furniahed
Sony Color Monitor	1 Customer Furnished
Sony LDP-1000A	1 Customer Furniahed
Sony LDP-1000A	1 Furnished
DDI Interface Unit	1 Furnished
Wico Joystick	1 Furnished
Power Cord	1 Furnished
RS 232 Cable (1-406A-1)	1 Furnished
RS 232 Cable (1-406A-2)	2 Furnished
Video Cable (1-406A-4)	7 Furniahed
ATR Maintenance Manual	1 Furnished
ATR Instructor'a Guide	1 Furnished
ATR Videodisc	2 Furnished
ATR Software Diskette	1 Furniahed

Table 4-1. LIST OF MATERIALS

4.2 Setup

Figure 4-1 shows one possible arrangement for the ATR System components. Note that the Sony players cannot be stacked. Follow the instructions on the players for unlocking the heads when setting them up. Set the EXT CPU switch on each videodisc player to ON and the SC/SYNC switch to EXT. Connect the RS 232 cable and the three video cables to each player and to the appropriate connectors on the ATR Interface Unit. Connect the RS 232 cable from the MicroTICCIT terminal and the video cable from the monitor to the Interface Unit. Plug in all of the power cords. Connect the joystick to the front of the Interface Unit. Follow the instructions in Chapter 5 for powering up and operating the ATR System.

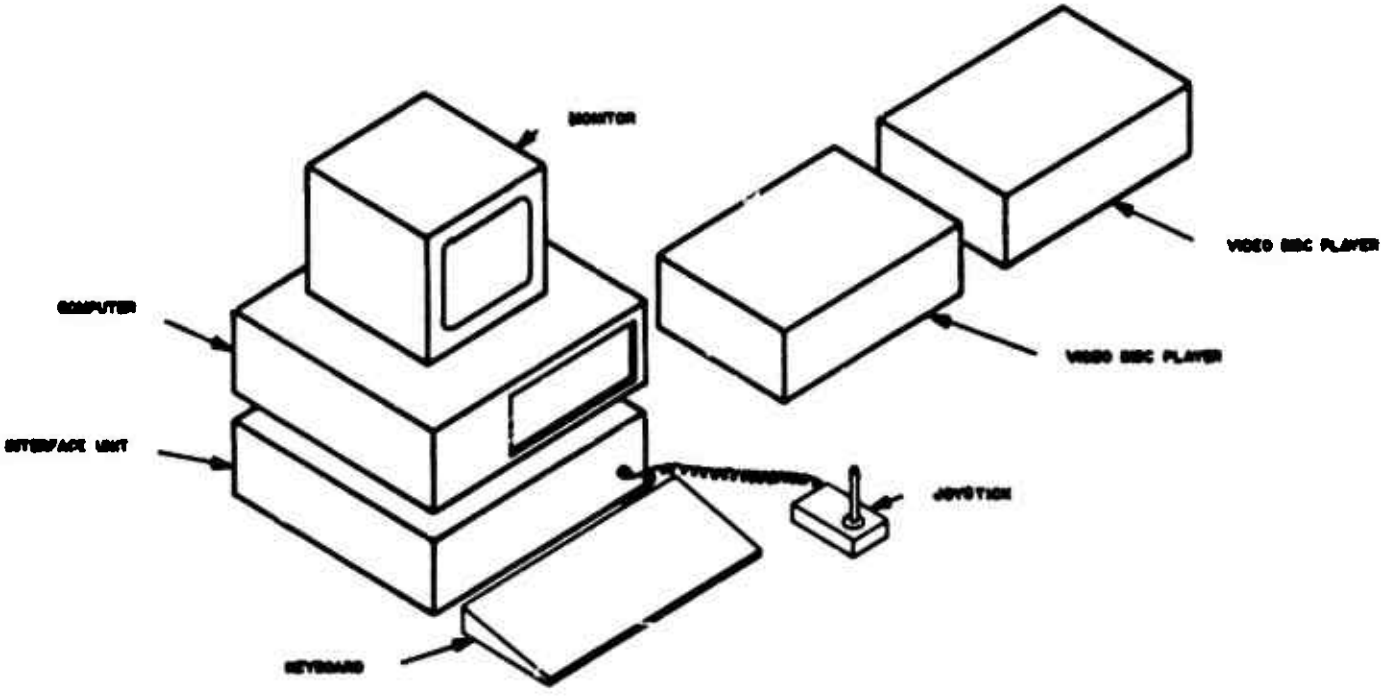


Figure 4-1. ATR SYSTEM LAYOUT

4.3 Packing Instructions

If it should be necessary to ship either the Interface Unit or a videodisc player, use the original box if it is available. Otherwise, any strong box large enough to hold the component and ample packing material for cushioning will do. Be sure to follow the manufacturer's instructions for locking the heads on the videodisc players. If a component is being sent to DDI for maintenance, be sure to include the cables for that component so that they can be tested too.

5.0 OPERATION

The purpose of this chapter is to provide a brief description of the operating procedures for the ATR System. This is not intended to be a comprehensive user's guide, but rather a basic "how to" on system startup and checkout. If more information is needed refer to the ATR Instructor's Guide.

5.1 System Startup

The startup procedure for the ATR System is straightforward. First the monitor, terminal, interface unit and the disc players are turned on. There is no special sequence to turning on the various components. Load the ATR videodiscs into the players, shiny side down. The videodiscs are identical so either videodisc can go in either player. Load the ATR diskette into the left drive on the MicroTICCIT terminal. There should be a series of clicks, and a screen with white lettering on a blue background will appear. From this point follow the instructions on the screen. Figure 5-1 shows the sequence. A good system checkout is provided by the Free Travel menu choice. Roam about on the terrain to check joystick commands, player operation and Interface Unit performance.

5.2 System Shutdown

When ready to shut the system down, simply halt the program to get back to the A> prompt and turn off the ATR hardware. The videodiscs and the floppy disk may be left in place while the power is off.

(Power up the subassemblies, load the videodiscs and the software diskette)

(A blue screen with white border appears)

ADVANCED TERRAIN REPRESENTATION

⋮

Press button on joystick to continue

(Press joystick button)

(A menu appears)

Free Travel

⋮

Shut Down ATR System

(Choose Free Travel for system checkout)

(Terrain image appears)

(Use joystick to move about on terrain)

Figure 5-1. SAMPLE ATR DIALOGUE

6.0 MAINTENANCE

The purpose of this chapter is to give the ATR user guidance on how to keep the system working and what to do if it is not working. Procedures are given for determining which sub-assembly is defective. Most of the components making up the ATR system are not user serviceable. Any repairs must be done by qualified personnel.

6.1 Periodic Maintenance

The ATR system does not require any periodic maintenance. Simply keep the various subassemblies clean and avoid spilling food or drink into anything. If a subassembly should get wet, contact DDI for advice before turning it on.

6.2 Tests and Diagnostics

Table 6-1 lists symptoms and their possible causes. This is by no means a comprehensive list. If the ATR System is displaying novel symptoms, first check all of the cable connections and switch settings. Then check to make sure the proper videodiscs and floppy diskettes are loaded. If everything is connected properly, and the symptoms haven't disappeared then it will be necessary to isolate which subassembly is at fault.

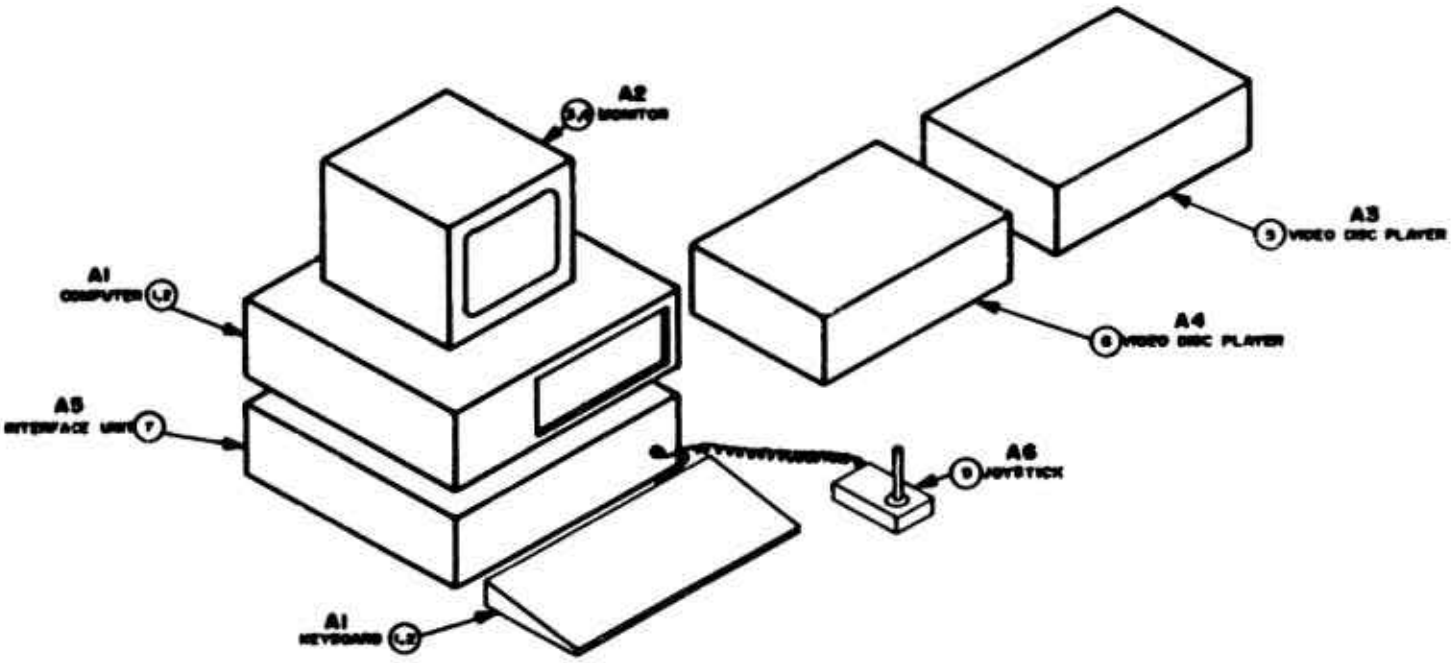
SYMPTOM	POSSIBLE CAUSES
1. ATR System is dead	1. Power Cords unplugged. 2. Breaker for outlet has tripped.
2. MicroTICCIT Terminal won't boot	1. Unit is unplugged. 2. Unit is turned off. 3. Diskette is inserted incorrectly or missing. 4. Terminal is faulty.
3. Terminal appears to boot but no display appears on the monitor.	1. Monitor is unplugged. 2. Monitor is turned off. 3. Cables are incorrectly connected. 4. Diskette is inserted incorrectly or missing. 5. Monitor is faulty. 6. Terminal is faulty.
4. Terminal cannot talk to videodisc players	1. Players are unplugged. 2. Players are turned off. 3. Videodiscs are not loaded or loaded incorrectly. 4. Interface Unit is turned off. 5. External CPU switches on players are set incorrectly. 6. Cables are incorrectly connected. 7. Player is faulty. 8. Interface Unit is faulty.
5. Terrain images appear but lack color or stability	1. SYNC and SC cables are incorrectly connected. 2. External Sync switch on players are set incorrectly. 3. Player is faulty. 4. Interface Unit is faulty. 5. Terminal is faulty.
6. Joystick commands not accepted by terminal	1. Joystick is unplugged. 2. Interface Unit is turned off. 3. Interface Unit faulty. 4. Joystick is faulty.
7. Random Static occurs in Terrain imagery.	1. Video/Graphic board in terminal is faulty.

Table 6-1. TROUBLESHOOTING GUIDE

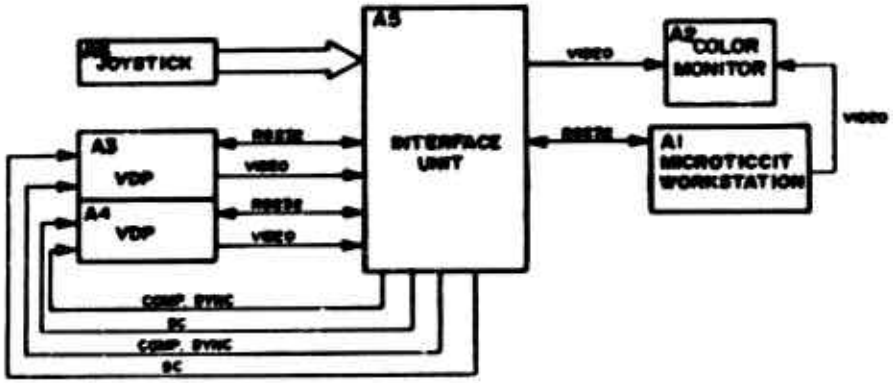
APPENDIX A

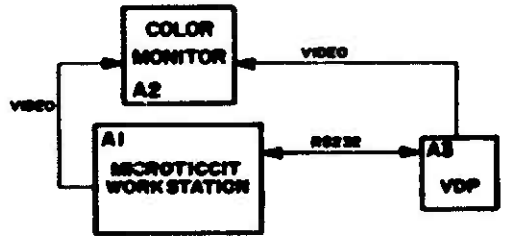
TECHNICAL DRAWINGS

INDEX TO APPENDIX A



CONTRACT NO 367-3		DECISIONS AND DESIGNS INC.
APPROVALS	DATE	
DRAWN H. MONTERA	4/25/89	ASSEMBLY,(ATR) SYSTEM
CHECKED		ADVANCED TERRAIN REPRESENTATION
		SIZE B / CODE IDENT NO / DRAWING NO. I-423 B
		SCALE / SHEET 1 OF 2





ORIGINAL TICCIT CONFIGURATION

NOTE: UPON POWER DOWN INTERFACE UNIT SWITCHES TO ORIGINAL TICCIT CONFIGURATION

CONTRACT NO. 207-3		DECISIONS AND DESIGNS INC.
APPROVALS	DATE	
DRAWN B. BERTINA	1/15/75	ASSEMBLY (ATR/TICCIT) SYSTEM
CHECKED [signature]	5-9-75	
[signature]	5-9-75	
		SIZE B; CODE IDENT NO.; DRAWING NO. I-423 B
		SCALE; SHEET 2 OF 2

Electrical Reference

Block Diagram: 1-361B
Detailed Block Diagram: 1-359B

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

Date MAY 9, 1985
Approval Eliot Mac

PARTS LIST 1-423B Sheet 1 of 1

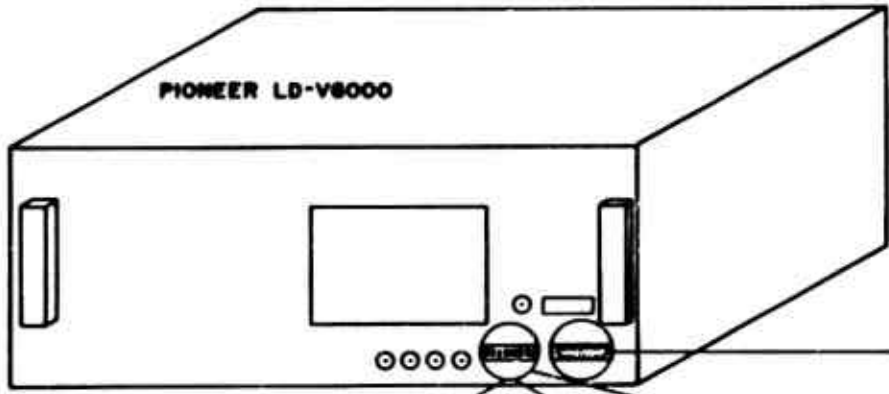
ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY	
		ASSEMBLY, ATR SYSTEM	DDI	TICCIT	1-423B-1	↓	
		ASSEMBLY, ATR SYSTEM	DDI	DDI	1-423B-2		↓
1	A1	COMPUTER SYSTEM	HAZELTINE	(CUSTOMER FURNISHED)		1	-
2	A1	COMPUTER SYSTEM	COMPAQ			-	1
3	A2	MONITOR, COLOR	SONY	(CUSTOMER FURNISHED)		1	-
4	A2	MONITOR, COLOR NTSC	SANYO	DMC-6013		-	1
5	A3, A4	PLAYER VIDEODISC	SONY	(CUSTOMER FURNISHED)		2	-
6	A3, A4	PLAYER VIDEODISC	PIONEER	LD-V6000		-	2
7	A5	ASSEMBLY INTERFACE UNIT	DDI	ATR	1-422B-1	1	1
8							
9	A6	JOYSTICK	WICO	50-0299		1	1
10							
11		ASSEMBLY CABLE RS-232	DDI	COMPUTER	1-406A-1	-	1
12		ASSEMBLY CABLE RS-232	DDI	LDP-1000A	1-406A-2	2	-
13		ASSEMBLY CABLE RS-232	DDI	LD-V6000	1-406A-3	-	2
14		ASSEMBLY CABLE VIDEO	DDI		1-406A-4	6	7

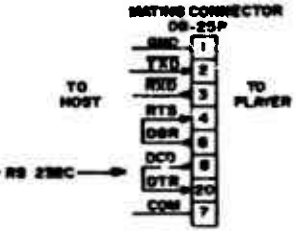
PIONEER FIELD MODIFICATION NO.
SI-F85007 PERFORMED AS OF 2/13/85

ALL VDP'S AJ OF ABOVE DATE
CONTAIN EPROM VER. 3.3

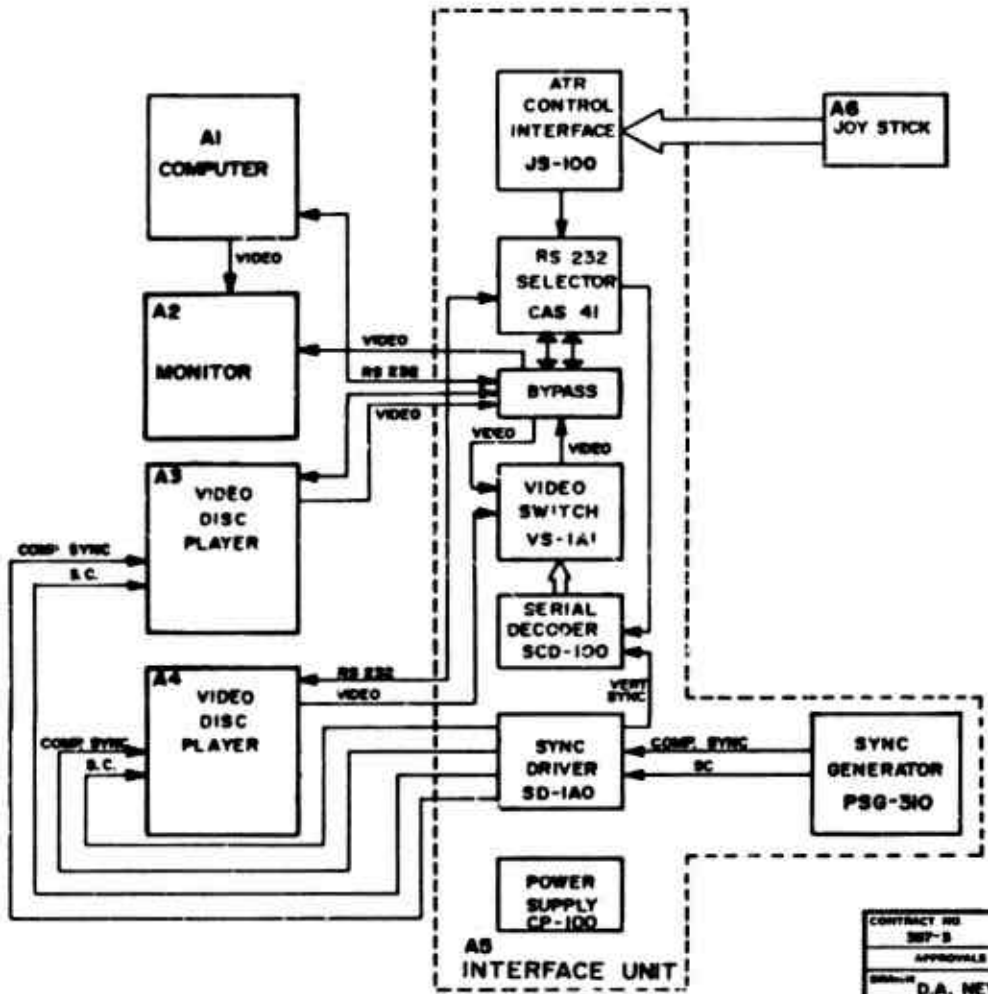
SWITCH 1,2,3 ARE
BAUD RATE SELECT

SWITCH 7 IS BIT SELECT
ON = 8 BIT DATA
OFF = 7 BIT DATA

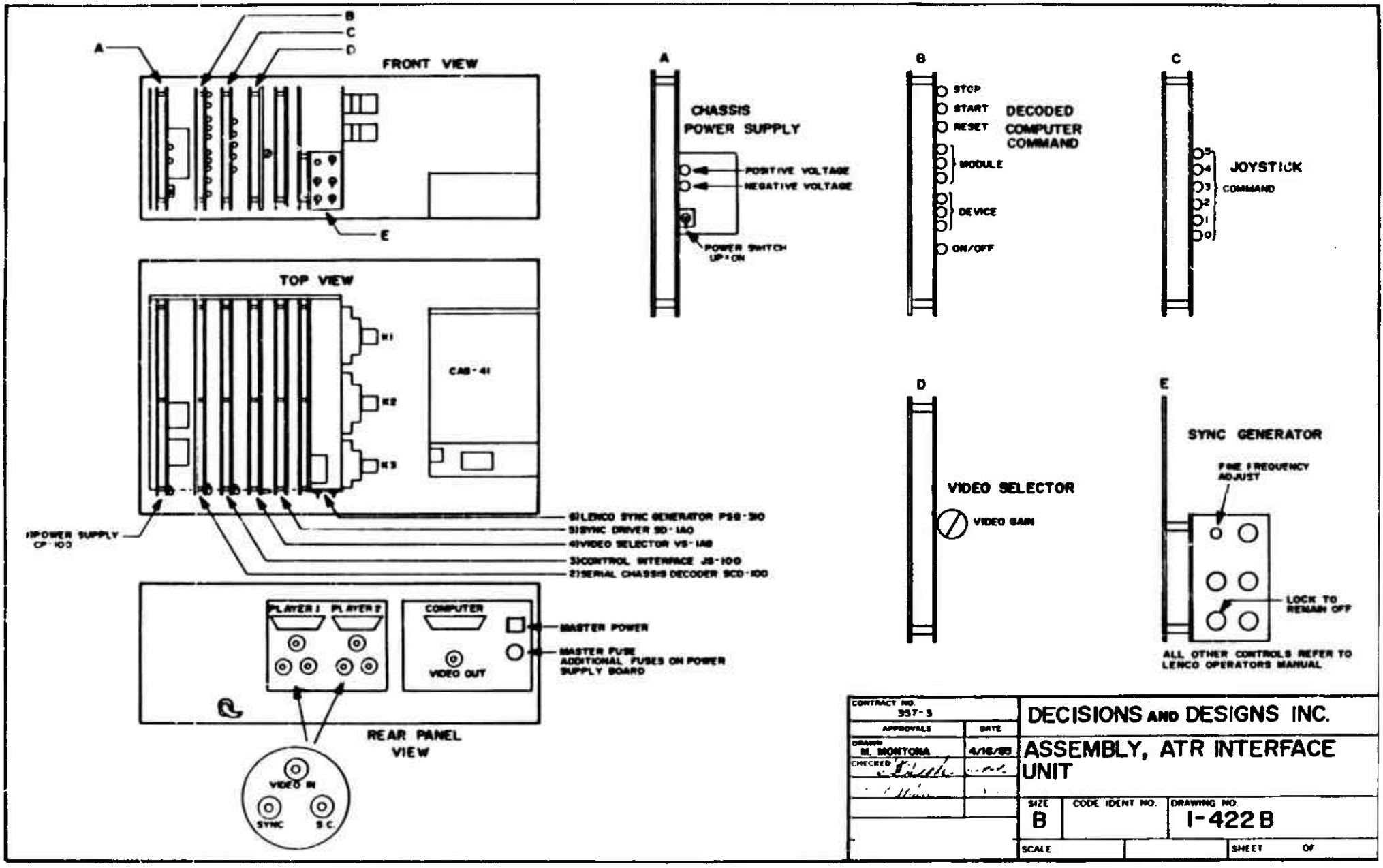




CONTRACT NO.		DECISIONS AND DESIGNS INC.
APPROVALS	DATE	
DRAWN M. MONTOYA	2/13/85	CONFIGURATION DIAGRAM
CHECKED	4-27-85	VIDEO DISC PLAYER
	5-29-85	PIONEER MODEL LD-V6000
		SIZE B / CODE IDENT NO. / DRAWING NO. I-404B
		SCALE / SHEET OF



CONTRACT NO 387-5		DECISIONS AND DESIGNS INC.
APPROVALS	DATE	
DRAWN D.A. NEVEU	12-18-84	BLOCK DIAGRAM
CHECKED	5-8-85	ATR, DETAILED
	5-9-85	
SIZE B	CODE IDENT NO.	DRAWING NO. 1-359 B
SCALE		SHEET OF



CONTRACT NO. 357-3		DECISIONS AND DESIGNS INC.
APPROVALS	DATE	
DRAWN M. MONTOYA	4/16/83	ASSEMBLY, ATR INTERFACE UNIT
CHECKED		
SIZE B	CODE IDENT NO.	DRAWING NO. I-422 B
SCALE		SHEET OF

Electrical Reference

DETAILED BLOCK DIAGRAM 1-359B

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

Date 18 MAY 1985
Approval

PARTS LIST 1-422B Sheet 1 of 2

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
		ASSEMBLY, ATR INTERFACE UNIT	DDI		1-422B-1	↓
1		MODULE CHASSIS	DDI	(MODIFIED)	1-260A	1
2	1	ASSEMBLY, CHASSIS POWER SUPPLY	DDI	CP-100	1-230C-1	1
3	2	ASSEMBLY, SERIAL CHASSIS DECODER	DDI	SCD-100	1-414C-1	1
4	3	ASSEMBLY, CONTROL INTERFACE	DDI	JS-100	1-425C-1	1
5	4	ASSEMBLY, VIDEO SELECTOR	DDI	VS-1A8	1-225C-1	1
6	5	ASSEMBLY, SYNC DRIVER	DDI	SD-1A0	1-418C-1	1
7	6	SYNC GENERATOR	LENCO	PSG-310		1
8		SHIELD PLATE, PC BOARD	DDI	(MODIFIED)	1-219B-1	1
9		MULTIPLEXIER, RS-232	WTI	CAS-41		1
10		SOCKET, RELAY	IDEC	RH-2B		3
11	K1-K3	RELAY, 12V DPDT	IDEC	RH2B-U		3
12		CONNECTOR, RS-232	TRW	DB-25S		3
13		COAX CONNECTOR, BNC	AMPHENOL	31-10		7
14		CONNECTOR, JOYSTICK	TRW	DB-9P		1

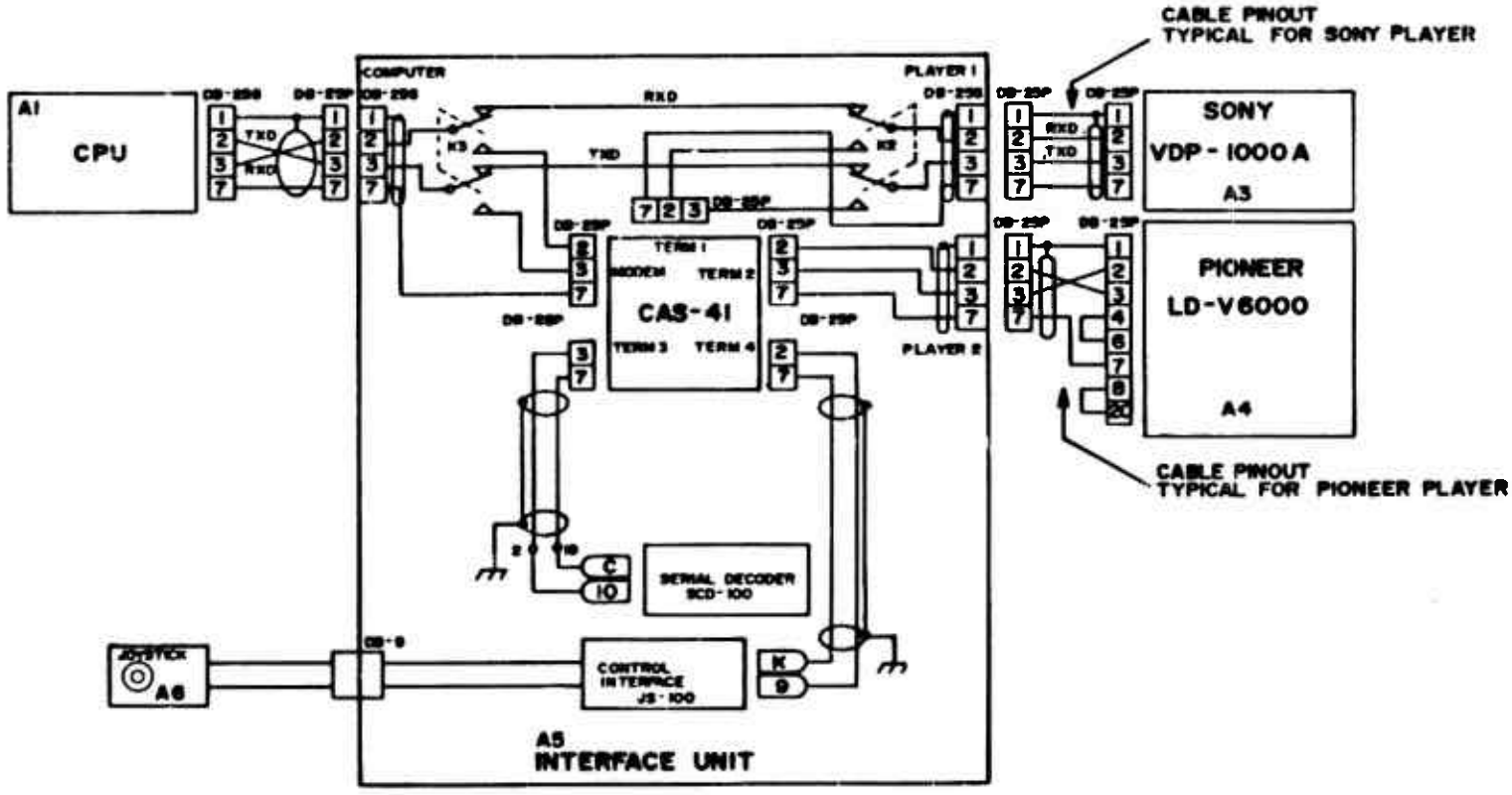
Electrical Reference

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

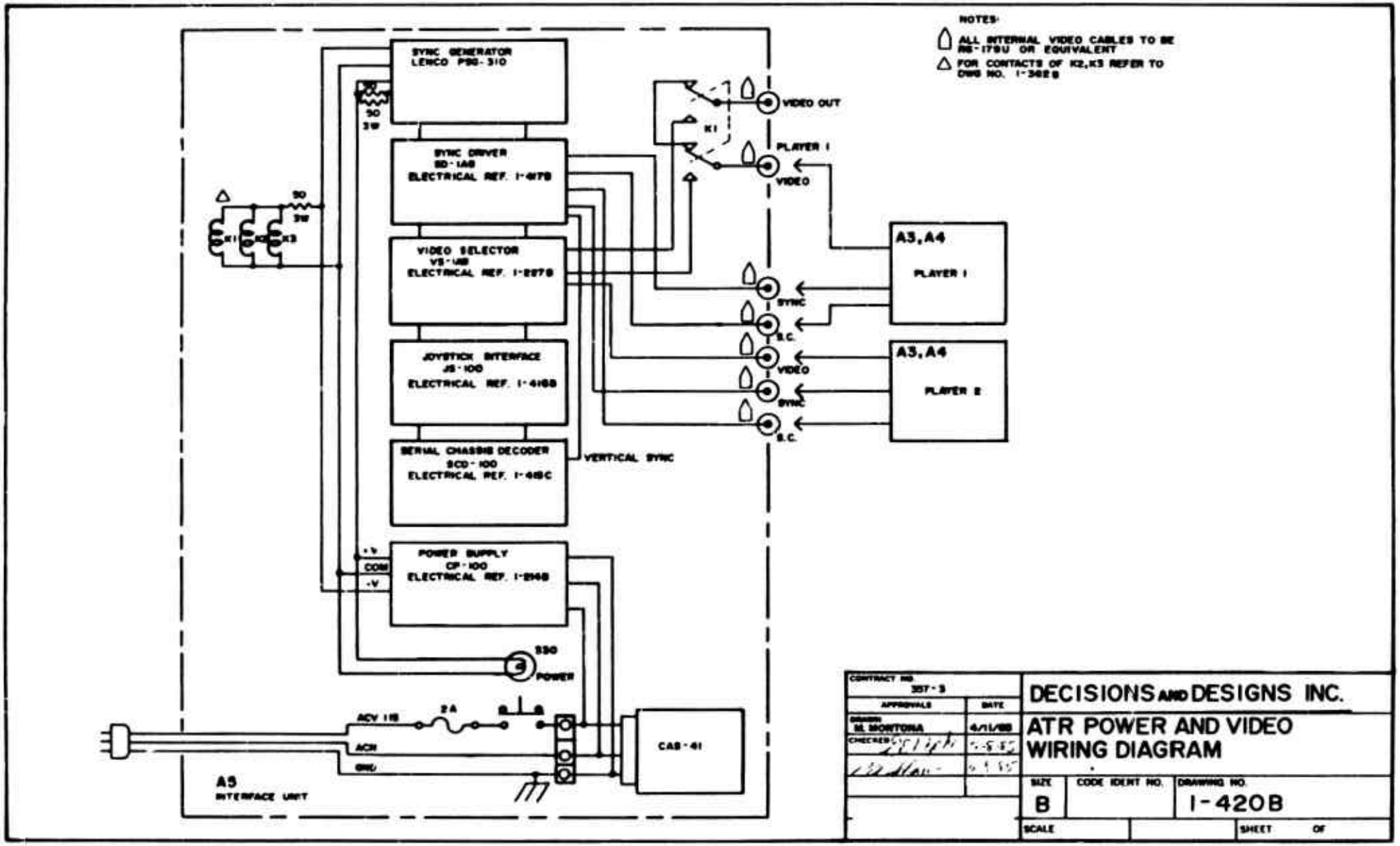
Date 8 MAY 1985
Approval

PARTS LIST 1-422B Sheet 2 of 2

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
15		FUSE HOLDER	LITTLEFUSE			1
16		FUSE 2A 3AG	LITTLEFUSE			1
17		POWER SWITCH	GC ELECTRONICS	35-484		1
18		CONNECTOR RS-232	TRW	DB-25P		5
19		CABINET, IBM PC	CPG	(MODIFIED)		1
20		PANNEL PLEXIGLASS	DDI	(SPECIAL)		1
21		PANNEL, REAR CONNECTOR	DDI	(SPECIAL)		1



CONTRACT NO 287-3		DECISIONS DESIGNS INC.
APPROVALS	DATE	
DRAWN M. MONTOYA	1/9/85	ATR RS 232 CABLE PINOUT DIAGRAM
CHECKED	5-8-85	
	5-8-85	
SIZE B	CODE IDENT NO.	DRAWING NO. 1-362 B
SCALE		SHEET OF


NOTES:
ALL INTERNAL VIDEO CABLES TO BE RG-179U OR EQUIVALENT
FOR CONTACTS OF K2,K3 REFER TO DWG NO. 1-3628
SYNC GENERATOR LENCO PSG-310
50 3W
SYNC DRIVER SD-148 ELECTRICAL REF. 1-4178
VIDEO SELECTOR VS-148 ELECTRICAL REF. 1-2278
JOYSTICK INTERFACE JS-100 ELECTRICAL REF. 1-4168
SERIAL CHASSIS DECODER SCD-100 ELECTRICAL REF. 1-4GC
POWER SUPPLY CP-100 ELECTRICAL REF. 1-2148
+V
COM
-V
K1
K2
K3
VIDEO OUT
PLAYER 1 VIDEO
SYNC
S.C.
VIDEO
SYNC
S.C.
A3,A4 PLAYER 1
A3,A4 PLAYER 2
VERTICAL SYNC
SSO POWER
ACV 115
2A
ACH
GND
CAB-41
A5 INTERFACE UNIT
DECISIONS AND DESIGNS INC.
ATR POWER AND VIDEO WIRING DIAGRAM
CONTRACT NO. 357-3
APPROVALS
DATE
DESIGN M. MONTONA 4/11/88
CHECKED
SIZE B
CODE IDENT NO.
DRAWING NO. 1-420B
SCALE
SHEET OF

Electrical Reference

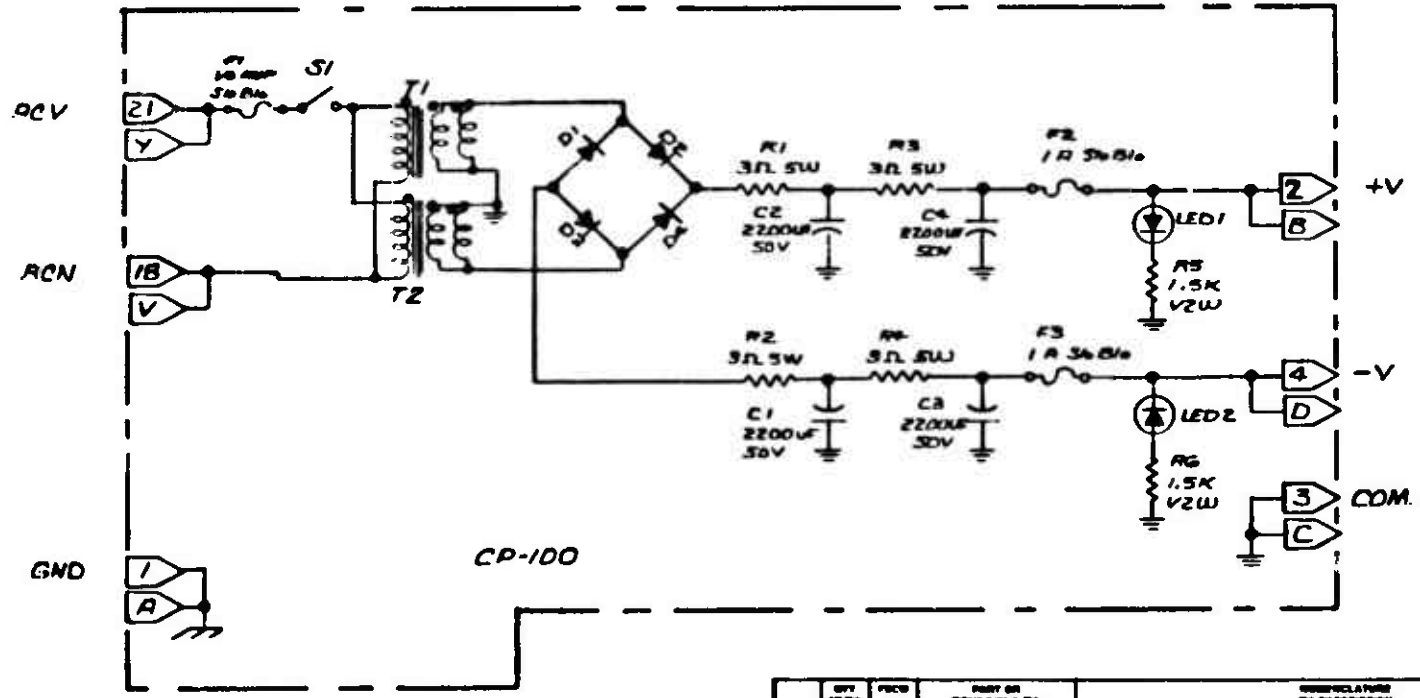
1-362B
1-420B

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

Date 5-8-85
Approval

PARTS LIST 1-406A Sheet 1 of 1

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
		ASSEMBLY, CABLE RS-232	DDI	COMPUTER	1-406A-1	↓
		ASSEMBLY, CABLE RS-232	DDI	LDP-1000A	1-406A-2	↓
		ASSEMBLY, CABLE RS-232	DDI	LD-V6000	1-406A-3	↓
		ASSEMBLY, CABLE COAX	DDI	VIDEO	1-406A-4	↓
1		CONNECTOR, BNC	AMPHENOL	68-175		- - - 2
2		CABLE, COAX	BELDEN	RG 59/U		- - - AR
3						
4		CONNECTOR RS-232	TRW	DB-25P		2 2 2 -
5						
6		CABLE, 4 COND SHLD	BELDEN	8723		AR AR AR -
7		COVER, CONNECTOR	AMP	2057181		2 2 2 -
8						
9						
10						
11						
12						



REVISIONS

REV	DESCRIPTION	DATE	APPROVED

PARTS LIST

QTY REQ'D	ITEM NO.	PART OR IDENTIFYING NO.	NOMENCLATURE OR DESCRIPTION	MATERIAL SPECIFICATION

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES
TOLERANCES ARE:
FRACTIONS ± DECIMALS .X ± .XX ± ANGLES ±

MATERIAL

FINISH

NEXT ASSY | USED ON

APPLICATION

DO NOT SCALE DRAWING

CONTRACT NO

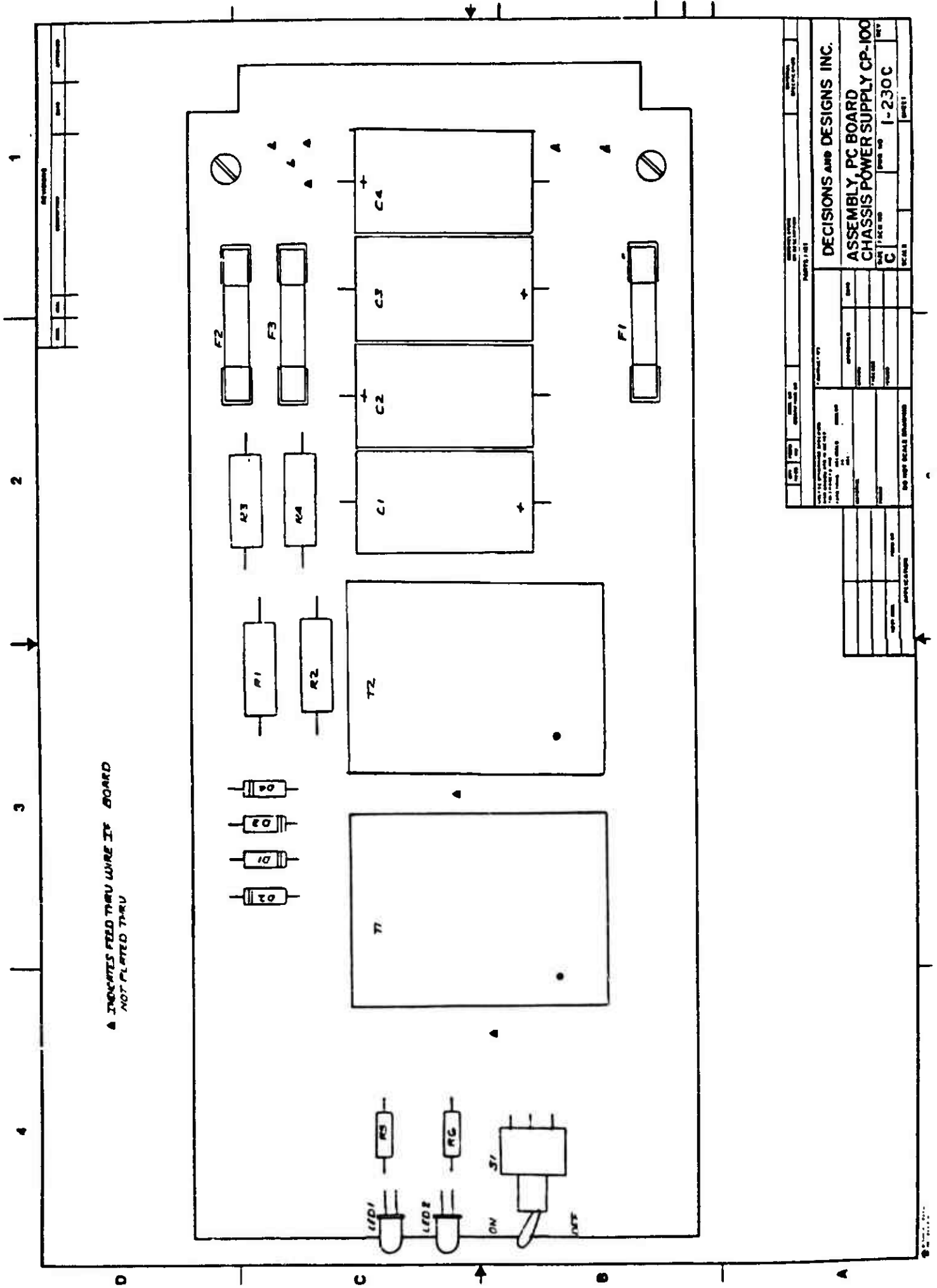
APPROVALS	DATE
DRAWN BEDICHE	11-3-81
CHECKED C KING	7-15-83
ISSUED	

DECISIONS AND DESIGNS INC

SCHEMATIC, CP-100
CHASSIS POWER SUPPLY

SIZE	FSCM NO.	DWG. NO.	REV.
B		1-2148	

SCALE | SHEET


DECISIONS AND DESIGNS INC.
ASSEMBLY, PC BOARD
CHASSIS POWER SUPPLY CP-100
C
1-230C
C4
C3
C2
C1
F1
F2
F3
R1
R2
R3
R4
T1
T2
D4
D3
D2
D1
LED1
LED2
ON
SET
INDICATES FEED THRU WIRE IF BOARD NOT PLATED THRU

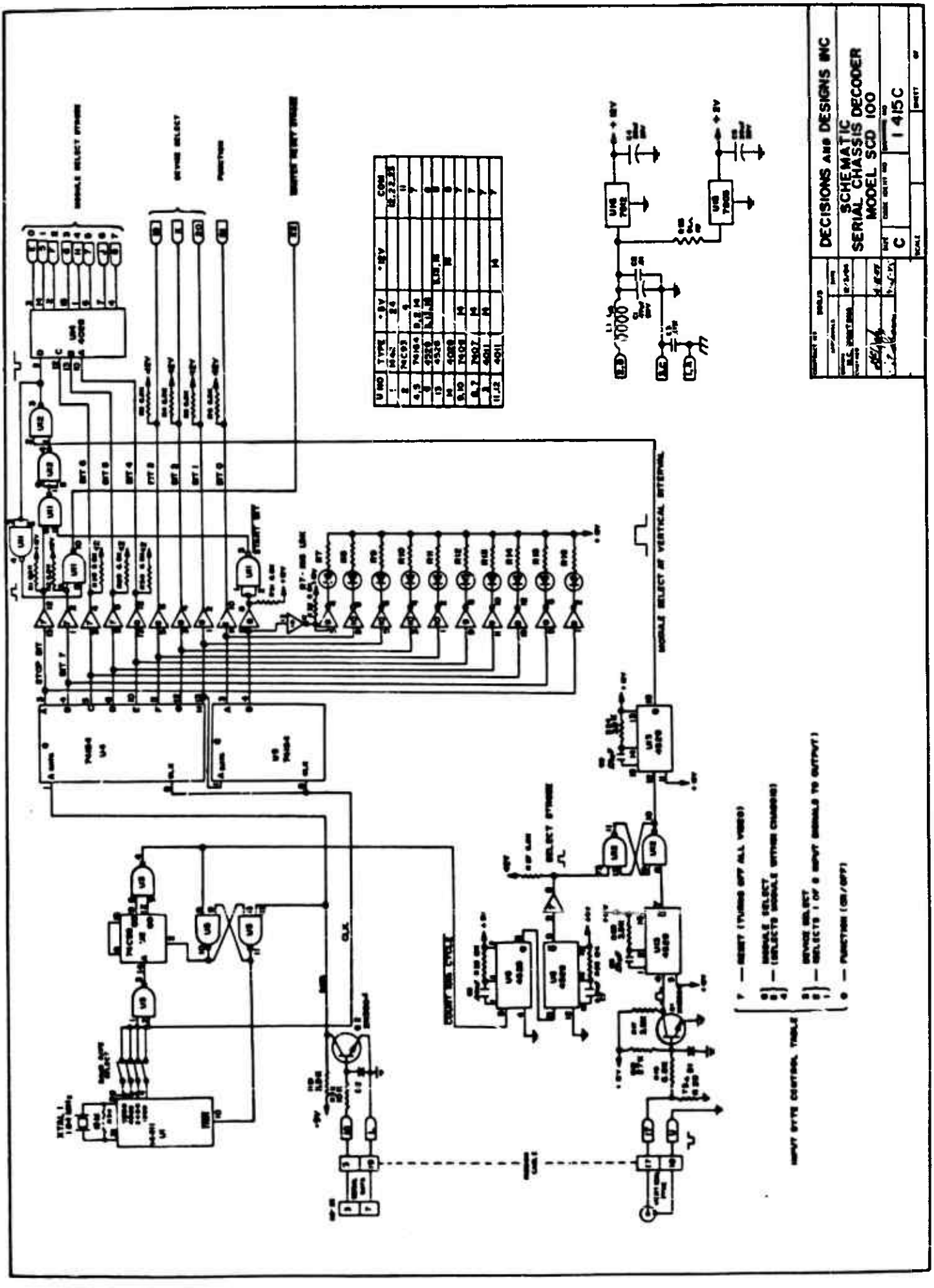
ELECTRICAL REFERENCE

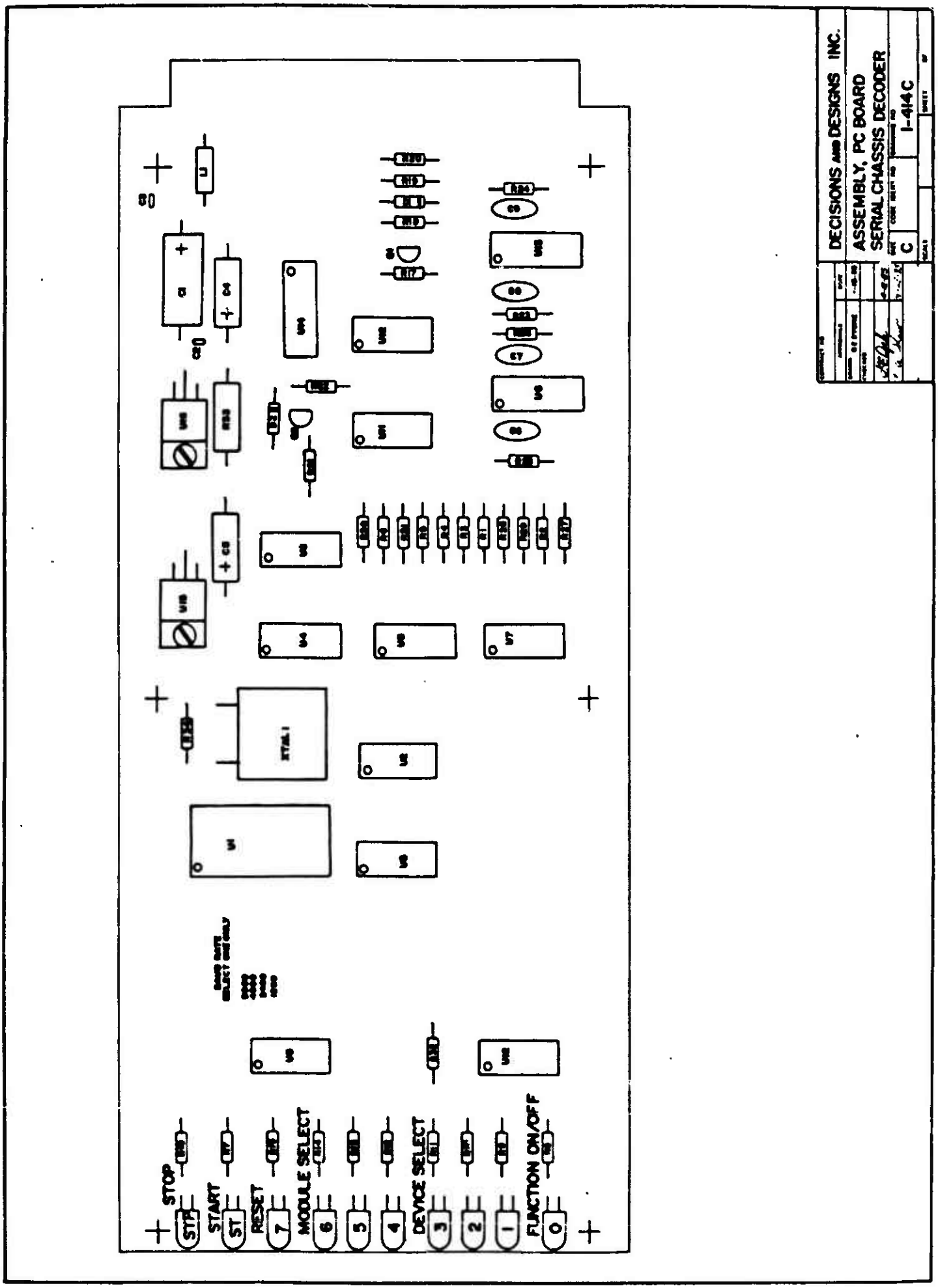
DECISIONS and DESIGNS, INC.
Suite 600, 8400 Westpark Drive, P.O. Box 907
McLean, Virginia 22101

Equip. Type Power Supply
Model No. CP-100
Serial No. _____
Date 12/3/81
Approval C. King

PARTS LIST 1-230C Sheet 1 of 1

ITEM	SYMBOL NO.	DESCRIPTION	MANUFACTURER	TYPE NO. MODEL NO.	DDI PART NO.	QUANTITY	REVISION
		ASSEMBLY PC BOARD	CHASSIS POWER SUPPLY	CP-100	1-230C-1	↓	
1		ARTWORK PC BOARD		CP-100	S1-030AWC	REF.	
2							
3							
4	T1, T2	TRANSFORMER, POWER	SIGNAL	PC34-700		2	
5		BRACKET, TRANSFORMER	SIGNAL	24-BR		2	
6							
7							
8	D1 - D4	DIODE, RECTIFIER	GE	IN4005		4	
9	LED1, 2	DIODE, LIGHT EMITTING		NSL5053		2	
10							
11	C1 - C4	CAPACITOR, 2200uf 50V	PANASONIC	ECEB1HV222S		4	
12	R1 - R4	RESISTOR, 3 ohm 5 watt	OHMITE	4542		4	
13							
14	R5, R6	RESISTOR, 1.5K ½ watt	IRC			2	
15							
16	F1	FUSE, ¼A SLO BLO	LITTLEFUSE	313.250		1	
17	F2, F3	FUSE, 1A SLO BLO	LITTLEFUSE	313001		2	
18		CLIP, FUSE HOLDING	LITTLEFUSE	102068		6	
19							
20	S1	SWITCH SPDT	ALCO	MTM-106D-RA		1	


DECISIONS AND DESIGNS INC
SCHEMATIC
SERIAL CHASSIS DECODER
MODEL SCD 100
1415C
C


DECISIONS and DESIGNS INC.
ASSEMBLY, PC BOARD
SERIAL CHASSIS DECODER
C
1-414C
STOP
START
RESET
MODULE SELECT
DEVICE SELECT
FUNCTION ON/OFF
STR
ST
7
6
5
4
3
2
1
0

Electrical Reference

1-415C

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

Date 4/1/85
Approval

PARTS LIST 1-414C Sheet 1 of 3

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
		ASSEMBLY, SERIAL CHASIS DECODER	DDI	SCD-100	1-414C-1	↓
1		ARTWORK PC BOARD		SCD-100	S1-414AWC	REF.
2		SHIELD PLATE PC BOARD	DDI		1-219B-1	1
3		SPACER 4/40 x 7/16"	SMITH	8829		6
4	U1	INTEGRATED CIRCUIT	MOTOROLA	MC14411P		1
5		SOCKET IC 24 PIN	TI	C8524		1
6	U2	INTEGRATED CIRCUIT	NATIONAL	MM74C93N		1
7	U3, U11, U12	INTEGRATED CIRCUIT	RCA	CD4011BE		3
8	U4, U5	INTEGRATED CIRCUIT	NATIONAL	MM74C164N		2
9	U6, U13	INTEGRATED CIRCUIT	NATIONAL	4528N		2
10	U7, U8	INTECRATED CIRCUIT	TI	SN7407N		2
11	U9, U10	INTEGRATED CIRCUIT	NATIONAL	7406N		2
12		SOCKET IC 14 PIN	TI	C8514		10
13		SOCKET IC 16 PIN	TI	C8516		3

Electrical Reference

1-415C

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

PARTS LIST 1-414C Sheet 2 of 3

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
14	U15	REGULATOR	TI	7805		1
15	U16	REGULATOR	TI	7812		1
16	D1, D2	DIODE, SIGNAL	GE	IN4148		2
17	LED 0 - LED STP	DIODE, LIGHT EMITTING	NSL	NSL5053		10
18	C1	CAPACITOR 47mFd 50V	PANASONIC	ECE-B4V470S		1
19	C2, .001	CAPACITOR .01mFd DISK	KEMET	C320C103MG5EA		2
20	C4, C5	CAPACITOR 22mFd 16V	PANASONIC	ECEBHV220S		
21	C6, C7	CAPACITOR .001mFd DISK	KEMET	C320C102MG5EA		2
22	C8, C9	CAPACITOR .01mFd DISK	KEMET	C320C103MG5EA		2
23	R1 - R6, R19, R27 - R32	RESISTOR 6.8k ¼W	IRC			12
24	R7 - R16	RESISTOR 1.5k ¼W	IRC			10
25	R17, R21, R23, R24	RESISTOR 3.9k ¼W	IRC			4
26	R22	RESISTOR 10k ¼W	IRC			1
27	R20	RESISTOR 75 ¼W	IRC			1
28	R18	RESISTOR 27k ¼W	IRC			1

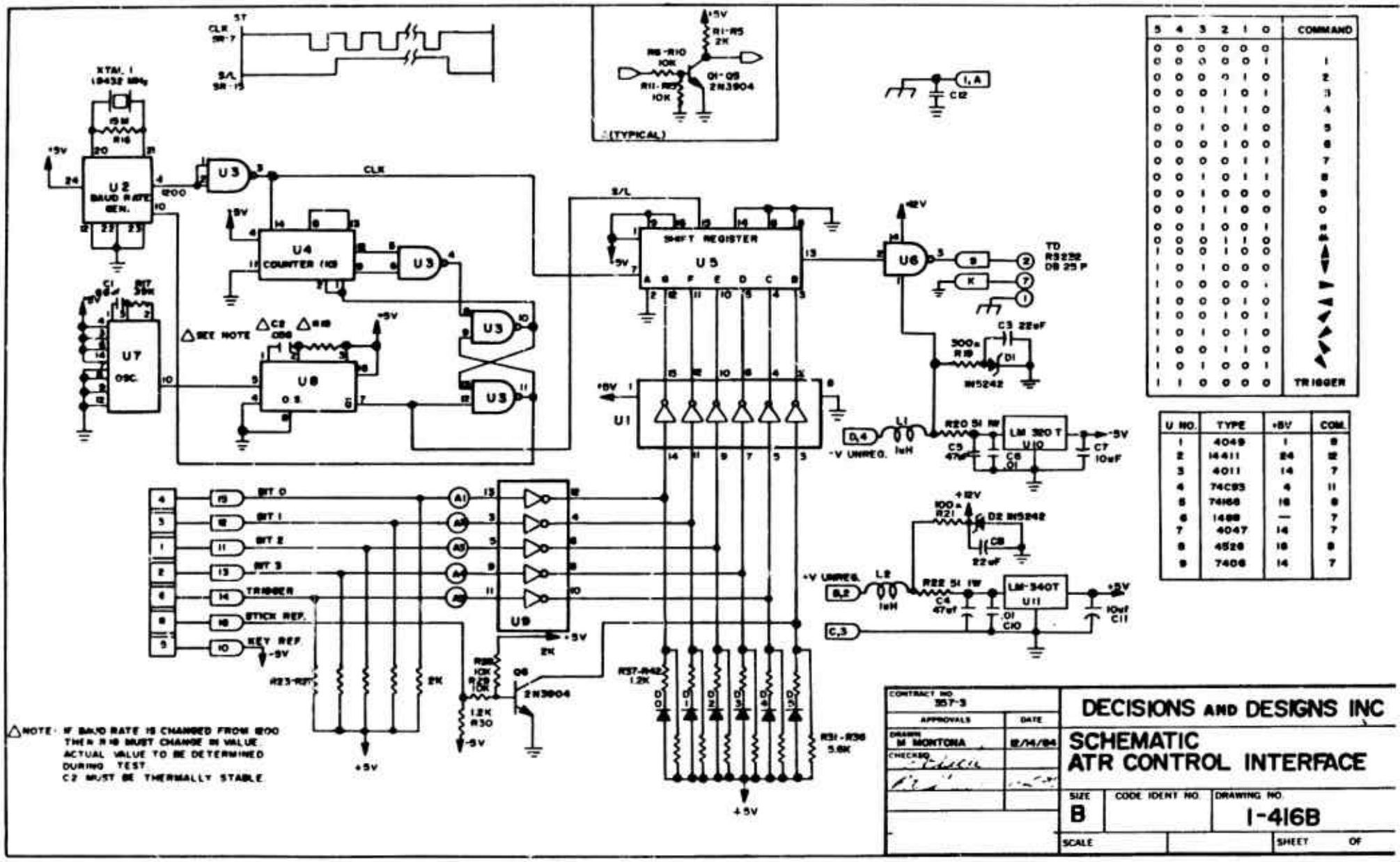
Electrical Reference

1-415C

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

PARTS LIST 1-414C Sheet 3 of 3

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
29	R25, R26	RESISTOR 12k ¼W	IRC			2
30	R33	RESISTOR 51 1W	IRC			1
31	R34	RESISTOR 15M ¼W	IRC			1
32	Q1,Q2	TRANSISTOR	MOTOROLA	2N3904		2
33	XTAL 1	CRYSTAL 1.8432 MHz	ITC			1
34	L1	CHOKE 1mH	MOUSER	43LS106		1
35	U14	INTEGRATED CIRCUIT	NATIONAL	4028CN		1

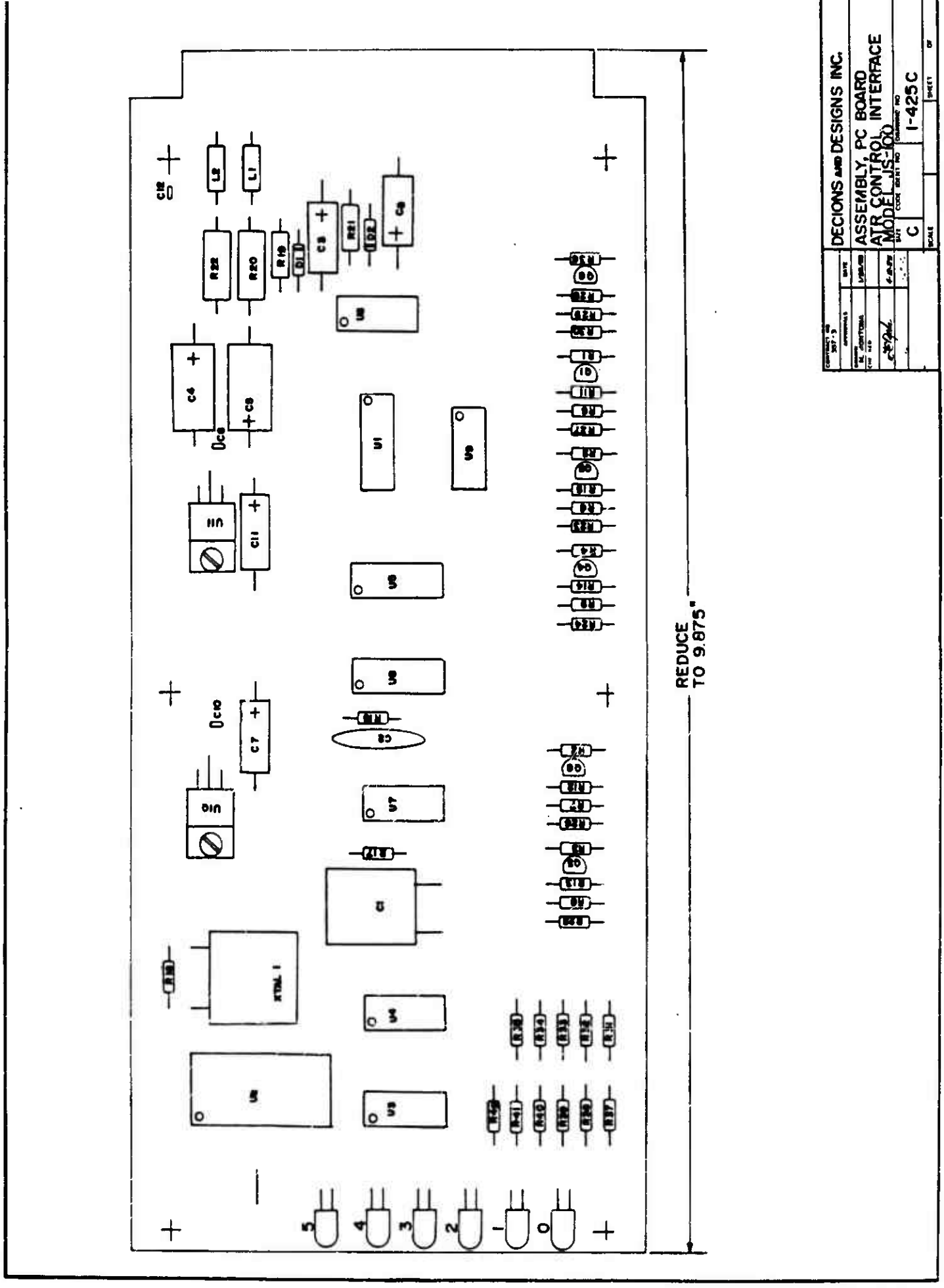


5	4	3	2	1	0	COMMAND
0	0	0	0	0	0	
0	0	0	0	0	1	1
0	0	0	0	1	0	2
0	0	0	0	1	1	3
0	0	0	1	0	0	4
0	0	0	1	0	1	5
0	0	0	1	1	0	6
0	0	0	1	1	1	7
0	0	1	0	0	0	8
0	0	1	0	0	1	9
0	0	1	0	1	0	0
0	0	1	0	1	1	←
0	0	1	1	0	0	→
0	0	1	1	0	1	↑
0	0	1	1	1	0	↓
0	0	1	1	1	1	TRIGGER

U NO.	TYPE	+5V	COM.
1	4049	1	8
2	14411	24	12
3	4011	14	7
4	74C93	4	11
5	74166	16	8
6	1488	—	7
7	4047	14	7
8	4528	16	8
9	7406	14	7

△NOTE: IF BAUD RATE IS CHANGED FROM 1200 THEN R18 MUST CHANGE IN VALUE. ACTUAL VALUE TO BE DETERMINED DURING TEST. C2 MUST BE THERMALLY STABLE.

CONTRACT NO. 357-3		DECISIONS AND DESIGNS INC
APPROVALS	DATE	SCHEMATIC ATR CONTROL INTERFACE
DRAWN M MONTONA	12/14/84	
CHECKED		
SIZE B	CODE IDENT NO.	DRAWING NO. 1-416B
SCALE		SHEET OF


DECIONS AND DESIGNS INC.
ASSEMBLY, PC BOARD
ATR CONTROL INTERFACE
MODEL US-100
C
1-425C
REDUCE
TO 9.875"

Electrical Reference

Schematic 1-416B
Joystick Schematic 1-363B (REF)

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

Date 4/1/85
Approval

PARTS LIST 1-425C Sheet 1 of 3

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
		ASSEMBLY, ATR CONTROL INTERFACE	DDI	JS-100	1-425C-1	
1		ARTWORK PC BOARD		JS-100	S1-415AWC	REF.
2		SHIELD PLATE PC BOARD	DDI		1-219B-1	1
3		SPACER 4/40 x 7/16"	SMITH	8829		6
4	U1	INTEGRATED CIRCUIT	NATIONAL	CD4049CN		1
5	U2	INTEGRATED CIRCUIT	MOTOROLA	MC14411P		1
6	U3	INTECRATED CIRCUIT	RCA	CD4011BE		1
7	U4	INTECRATED CIRCUIT	NATIONAL	MM74C93N		1
8	U5	INTECRATED CIRCUIT	NATIONAL	DM74166N		1
9	U6	INTECRATED CIRCUIT	NATIONAL	DS1488N		1
10	U7	INTEGRATED CIRCUIT	NATIONAL	4047N		1
11	U8	INTEGRATED CIRCUIT	NATIONAL	4528N		1
12	U9	INTECRATED CIRCUIT	NATIONAL	7406N		1
13		SOCKET 24 PIN	TI	C8524		1

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

PARTS LIST 1-425C Sheet 2 of 3

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
14		SOCKET 14 PIN	TI	C8514		5
15		SOCKET 16 PIN	TI	C8516		3
16	U10	REGULATOR	NATIONAL	LM-320T -5		1
17	U11	REGULATOR	NATIONAL	LM-340T 5		1
18	D1, D2	DIODE, ZENER	GE	IN5242		2
19	D0-D5	DIODE, LIGHT EMITTING	NSL	NSL5053		6
20	C1	CAPACITOR .68m Fd	TRACON	2A684K		1
21	C2	CAPACITOR - TBD				1
22	C3, 7, 8, 11	CAPACITOR 22mFd 16V	PANASONIC	ECEBHV220S		4
23	C4-5	CAPACITOR 47mFd 50V	PANASONIC	ECEBHV470S		2
24	C6, C10	CAPACITOR .01mFd Disk	KEMET	C320C103M65EA		2
25	R1-5, R23-27	RESISTOR 2k ¼W	IRC			10
26	R6-15, R28-29	RESISTOR 10k ¼W	IRC			12
27	R16	RESISTOR 15M ¼W	IRC			1
28	R17	RESISTOR 39k ¼W	IRC			1
29	R18	RESISTOR - TBD				1
30	R19	RESISTOR 300	IRC			1

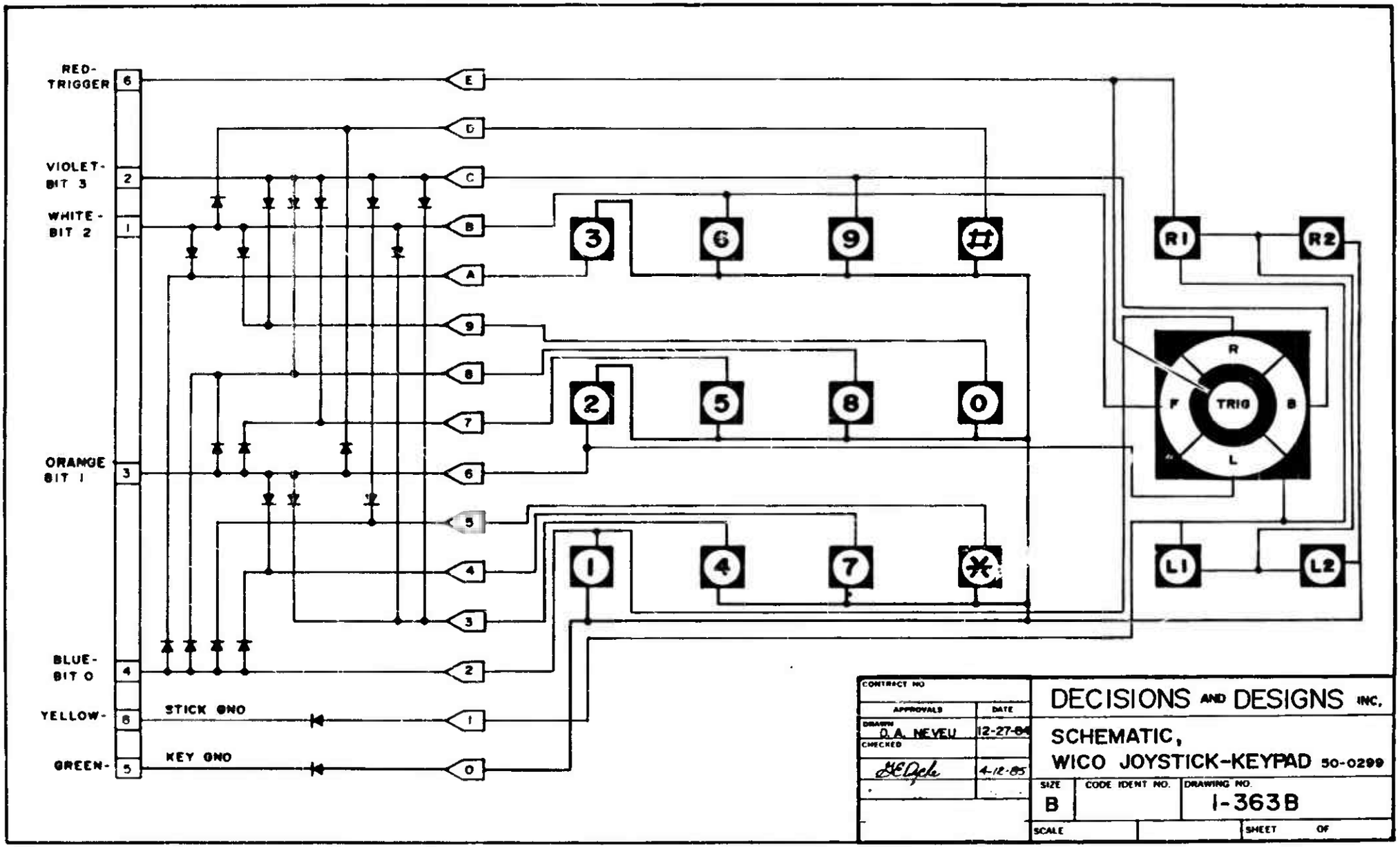
Electrical Reference

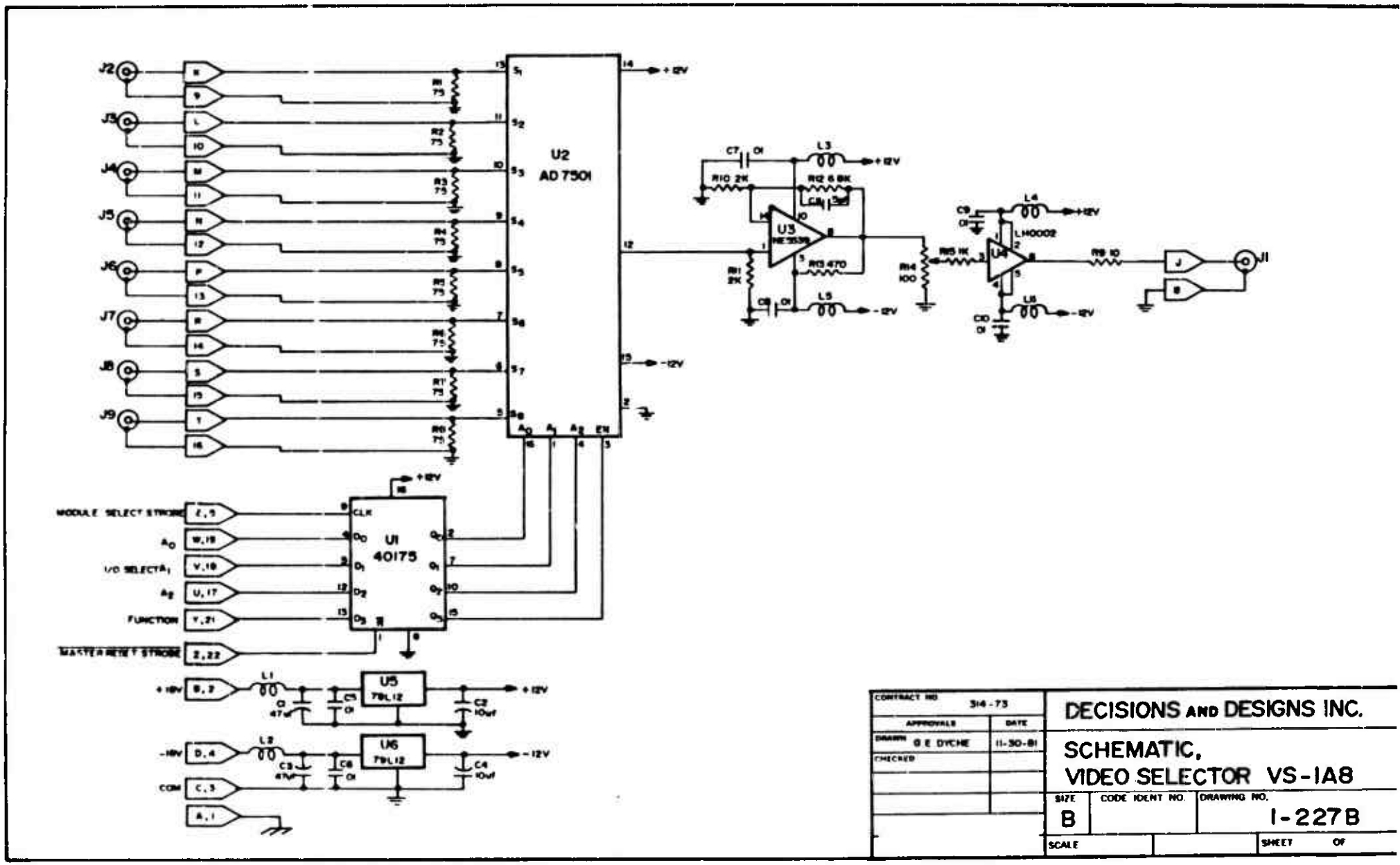
1-416B

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

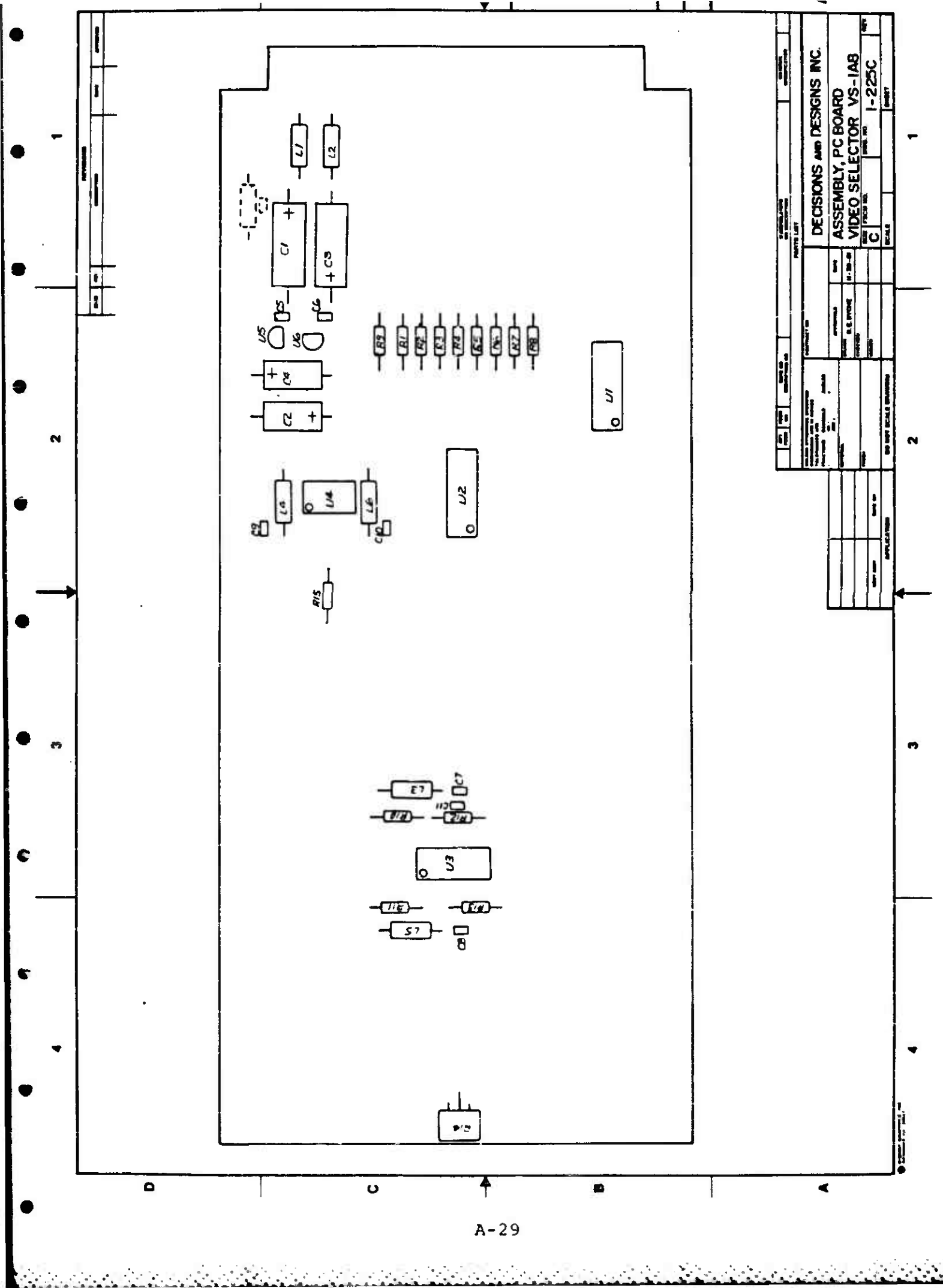
PARTS LIST 1-425C Sheet 3 of 3

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
31	R20, R22	RESISTOR 51 1W	IRC			2
32	R21	RESISTOR 500	IRC			1
33	R30, R37-42	RESISTOR 1.2k ¼W	IRC			7
34	R31-36	RESISTOR 5.6k ¼W	IRC			6
35	Q1-6	TRANSISTOR	MOTOROLA	2N3904		6
36	XTAL 1	CRYSTAL 1.8432 MHz	ITC			I
37	L1, L2	CHOKE 1 mH	MOUSER	43L5106		2
38	C12	CAPACITOR .001mFd	KEMET	C320C102M65EA		1


RED-
TRIGGER
6
VIOLET-
BIT 3
2
WHITE-
BIT 2
1
ORANGE
BIT 1
3
BLUE-
BIT 0
4
YELLOW-
8
GREEN-
5
STICK GND
KEY GND
E
D
C
B
A
9
8
7
6
5
4
3
2
1
0
3
6
9
#
2
5
8
0
1
4
7
*
R1
R2
L1
L2
R
F
TRIG
B
L
CONTRACT NO
APPROVALS
DATE
DRAWN
D.A. NEVEU
12-27-84
CHECKED
4-12-85
DECISIONS AND DESIGNS INC.
SCHEMATIC,
WICO JOYSTICK-KEYPAD 50-0299
SIZE
B
CODE IDENT NO.
DRAWING NO.
1-363B
SCALE
SHEET
OF



CONTRACT NO 314-73		DECISIONS AND DESIGNS INC.
APPROVALS	DATE	
DRAWN G E DYCHE	11-30-81	SCHEMATIC, VIDEO SELECTOR VS-1A8
CHECKED		
SIZE B	CODE IDENT NO.	DRAWING NO. 1-227B
SCALE		SHEET OF


DECISIONS AND DESIGNS INC.
ASSEMBLY, PC BOARD
VIDEO SELECTOR VS-IA8
C
1-225C
J1
J2
C1
C3
C5
C6
U5
U6
C4
C2
R9
R1
R2
R3
R4
R5
R7
R8
U1
U2
L4
U4
L6
C8
R15
E7
C7
C11
U3
S7
C8
P12

ELECTRICAL REFERENCE

1-227B

DECISIONS and DESIGNS, INC.
Suite 600, 8400 Westpark Drive, P.O. Box 907
McLean, Virginia 22101

Equip. Type VIDEO SELECTOR
Model No. VS-1A8
Serial No.
Date 12/15/81
Approval C. King

PARTS LIST 1-225C Sheet 1 of 2

ITEM	SYMBOL NO.	DESCRIPTION	MANUFACTURER	TYPE NO. MODEL NO.	DDI PART NO.	QUANTITY	REVISION
		ASSEMBLY, PC BOARD	VIDEO SELECTOR	VS1A8	1-225C-1	1	
1		ARTWORK, PC BOARD		VS1A8	S1-025AWC	REF.	
2							
3		SOCKET, IC 16PIN	TI	C9516		2	
4	U1	INTEGRATED CIRCUIT	NATIONAL	CD40175BCN		1	
5	U2	INTEGRATED CIRCUIT	ANALOG DEVICES	AD7501JN		1	
6		SOCKET, IC 14PIN	TI	C9514		1	
7	U3	INTEGRATED CIRCUIT	SIGNETICS	NE5539N		1	
8		SOCKET, IC 10PIN					
9	U4	INTEGRATED CIRCUIT	NATIONAL	LH0002CN		1	
10	U5	REGULATOR, PLUS 12 VOLTS	NATIONAL	78L12A		1	
11	U6	REGULATOR, MINUS 12 VOLTS	NATIONAL	79L12ACP		1	
12							
13							
14	C1, C3	CAPACITOR, 47uf 50V.	PANASONIC	ECEBHV470S		2	
15	C2, C4	CAPACITOR, 10uf 50V.	PANASONIC	ECEBHV100S		2	
16	C5, C10	CAPACITOR, .01uf DISK	KEMET	C320C103MG5EA		6	
17	C11	CAPACITOR, 5 pico farad	SPRAGUE	10TCCV50		1	
18							
19	R1 - R8	RESISTOR, 75 ohm ¼ watt	IRC			8	
20	R9	RESISTOR, 10 ohm ¼ watt	IRC			1	

ORIGINAL

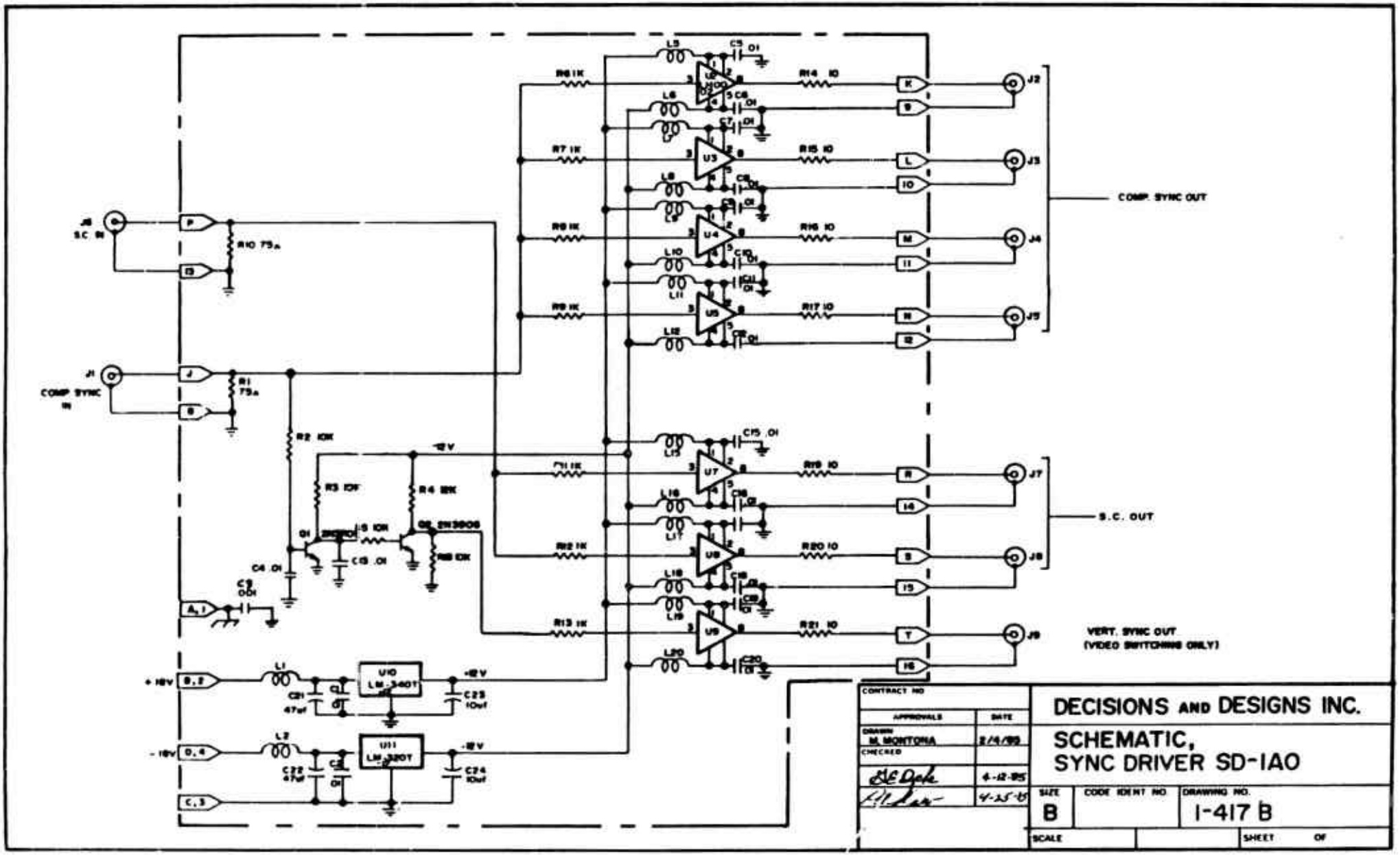
ELECTRICAL REFERENCE
1-227B

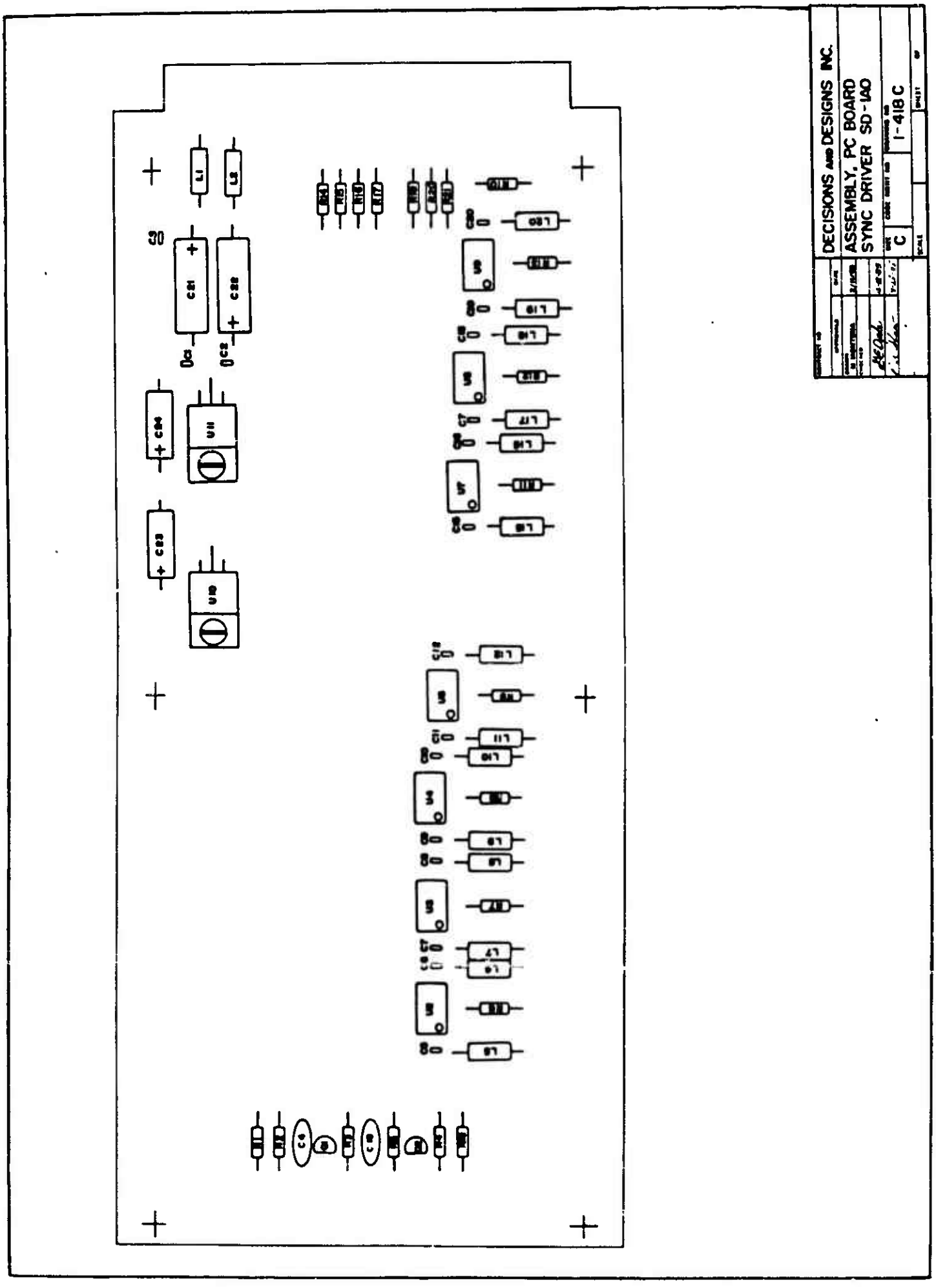
DECISIONS and DESIGNS, INC.
Suite 600, 8400 Westpark Drive, P.O. Box 907
McLean, Virginia 22101

VIDEO
Equip. Type SELECTOR
Model No. VS-1A8
Serial No.
Date 12/15/81
Approval C. King

PARTS LIST 1-225C Sheet 2 of 2

ITEM	SYMBOL NO.	DESCRIPTION	MANUFACTURER	TYPE NO. MODEL NO.	DDI PART NO.	QUANTITY	REVISION
21	R10, R11	RESISTOR, 2K ohm ¼ watt	IRC			1	
22	R12	RESISTOR, 6.8K ohm ¼ watt	IRC			1	
23	R13	RESISTOR, 470 ohm ¼ watt	IRC			1	
24	R14	POTENTIOMETER, 100 ohm	BECKMAN	91T-100		1	
25	R15	RESISTOR, 1K ¼ watt	IRC			1	
26							
27	L1 - L6	CHOKE, 1 micro henry	MOUSER	43LS106		1	
28							
29							
30							


COMP. SYNC OUT
S.C. OUT
VERT. SYNC OUT
(VIDEO SWITCHING ONLY)
+12V
-12V
+10V
-10V
DECISIONS AND DESIGNS INC.
SCHEMATIC,
SYNC DRIVER SD-1A0
M. MORTONA
2/9/89
4-12-89
4-25-89
SIZE
B
CODE IDENT NO.
DRAWING NO.
1-417 B
SCALE
SHEET OF


DECISIONS and DESIGNS INC.
ASSEMBLY, PC BOARD
SYNC DRIVER SD-1A0
C
1-418 C

Electrical Reference

1-417B

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

Date 4/1/85
Approval

PARTS LIST 1-418C Sheet 1 of 2

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
		ASSEMBLY, SYNC DRIVER	DDI	SD-1AO	1-418C-1	↓
1		ARTWORK, PC BOARD		SD-1AO	S1-418AWC	
2		SHIELD PLATE PC BOARD	DDI		1-219B-1	1
3		SPACER 4/40 x 7/16"	SMITH	8829		6
4	U1-U9	INTEGRATED CIRCUIT	NATIONAL	LH0002CN		7
5		SOCKET IC 10 PIN	TI	C8510		7
6	U10	REGULATOR	NATIONAL	LM-340T-12		1
7	U11	REGULATOR	NATIONAL	LM-320T-12		1
8	C1-C2, C5-20	CAPACITOR, .01mFd DISK	KEMET	C320C103MG5EA		17
9	C21, C22	CAPACITOR, 47mFd, 16V	PANASONIC	ECEBHV470S		2
10	C23, C24	CAPACITOR 22mFd 16V	PANASONIC	ECEBHV220S		2
11	R1, R10	RESISTOR, 75, 1/4W	IRC			2
12	R2, R3, R5, R18	RESISTOR, 10K, 1/4W	IRC			4

Electrical Reference

1-417B

DECISIONS AND DESIGNS, INC.
8400 Westpark Drive, Suite 600, P.O. Box 907
McLean, Virginia 22101

PARTS LIST 1-418C Sheet 2 of 2

ITEM	SYMBOL NO	DESCRIPTION	MANUFACTURER	TYPE/MODEL NOS	DDI PART NO	QUANTITY
13	R6-R13	RESISTOR, 1K, 1/4W	IRC			7
14	R4	RESISTOR, 12k, 1/4W	IRC			1
15	R14-R17, R19-R21	RESISTOR, 10, 1/4W	IRC			7
16	Q1, Q2	TRANSISTOR	MOTOROLA	2N3906		2
17	L1-L20	CHOKE, 1mH	MOUSER	43LS106		16
18	C3	CAPACITOR .001 mFd	KEMET	C320C102MG5EA		1

APPENDIX B

PROGRAM LISTING

```
/*****************************************************************/
/* Name:        atrdefs.h                                        */
/* Description:                                                  */
/*      These are the definitions of numerical constants for     */
/*      the ATR program configured with the following hardware:  */
/*          Hazeltine MicroTICCIT workstation                    */
/*              (IBM PC, SONY monitor, custom graphics card)     */
/*          Sony LDP-1000A videodisc player(s)                   */
/*          Western Telematic CAS-41 code activated switch       */
/*          DDI serial video switch and video driver             */
/*          WICO joystick with serial encoder/decoder            */
/*****************************************************************/

/* serial port parameters       */
#define MODE     0x83   /* 1.2kbps, 8 data/1 stop bit(s), no parity

/* commands for Western Telematic CAS-41        */
#define SELECT   0x00
#define VDBASE   0x41
#define VIDEO    0x44
#define JOYSTICK 0x48

/* commands for SONY LDP-1000A  */
#define CLEAR    0x56
#define DISABLE  0x51
#define DISPLAY  0x50
#define ENTER    0x40
#define NOENTRY  0xff
#define PLAY     0x62
#define REJECT   0x63
#define SEARCH   0x43
#define STATUS   0x67
#define STOP     0x4f

/* statuses returned from SONY LDP-1000A        */
#define ACK      0x0a
#define DONE     0x01
#define ERROR    0x02
#define NACK     0x0b
#define TIMEOUT  0x1000

/* commands for DDI serial video switch */
#define VIDBASE  0x11
#define BLANK    0x10

/* statuses returned from joystick      */
#define FIRE       -1
#define FWD      0xa4
#define LEFT     0xa2
#define REV      0xa8
#define RIGHT    0xa1
#define joywait(c)      while (joystat() != c);
#define keywait(c)      while (joystat() == c);

/* tone freqencies and lengths- */
#define ERRFREQ   125                   /* error tone - driving */
```

```
#define ERRTIME     2
#define ATTNFREQ  750                    /* get user's attention */
#define ATTNTIME    2

/* TICCIT color board registers and values      */
#define CRTCINDEX       0x300
#define CRTCDATA        0x301
#define INTERLACE           0
#define NONINTERLACE        1

#define VIDEOCTRL       0x302
#define EXTSYNC          0x48
#define INTSYNC          0x44

/* window and cursor macros, attributes */
#define MSGLN      24
#define NORMAL   0x17           /* white on blue attribute      */
#define REVIDEO  0x71           /* blue on white attribute      */
#define BRIGHT   0x13           /* cyan on blue attribute       */
#define clearw(w,a)     pcvsu(w,0,a);curs_off()
#define clearl(r,c,n,a) pcvscp(r,c);pcvwca(n,' ',a);curs_off()
#define cursor(r,c)     pcvscp(r,c);curs_on()

/* structure definition for route log entry     */
struct logentry
    {
    int point;          /* one character leg designation        */
    float odom_rdg;     /* current odometer reading             */
    float m_dist;       /* measured distance on map             */
    float a_dist;       /* actual distance (from odometer)      */
    int mag_azmth;      /* measured magnetic azimuth            */
    int declin;         /* declination (from map margin)        */
    int grd_azmth;      /* corrected grid azimuth               */
    );

/* miscellaneous definitions    */
#define TRUE        1
#define FALSE       0
#define HALT        0
```

```
/*****************************************************************/
/* Name:        askpos                                           */
/* Description:                                                  */
/*      This function gets the user's position from the keypad,  */
/*      while also allowing the user to pivot around and travel  */
/*      up to 30 meters from the home position.  Pressing the    */
/*      white button on the joystick takes the user back to the  */
/*      home position.  It checks to see if the user's guesses   */
/*      are within an allowable margin of error, and keeps the   */
/*      closest guess to return to the calling routine.          */
/* Inputs:      probno - which problem is being attempted        */
/*              x,y,z - user's home position                     */
/* Outputs:     z - user's orientation (updated)                 */
/*              dist - the closest guess to the actual position  */
/*              returns ntries - the number of guesses taken     */
/* Side effects:                                                 */
/*      User can only enter coordinates from the home position.  */
/*****************************************************************/
#include "ctype.h"
#include "atrdefs.h"

char *prmpt[] =         /* prompt for user's position            */
{
"W H E R E   A R E   Y O U ?",
"",
"Enter the six-digit coordinate for your position",
"[      ]"
};

char w[4];                          /* window for user's answer  */
char mask[] = {0,0,79,3};           /* window for text on video  */

extern int illegal[58][65];         /* bit map of illegal frames */
extern char window[];               /* useable portion of screens*/
extern char player;                 /* currently active player   */
extern char nplayers;               /* number of active players  */

askpos(px,py,pz,dist)
int *px,*py,*pz;
float *dist;
{
    int i,c;                        /* index and loop counter    */
    int x,y,z;                      /* local copies of coordinates */
    int oldx,oldy,oldz;             /* coords of previous position */
    int sameframe;                  /* flag to show no change    */
    int found;                      /* frame found on disc?      */
    int ntries = 0;                 /* number of guesses by user */
    int ansgotten;                  /* user has pressed RETURN key */
    int done;                       /* user guessed correctly    */
    int stat;                       /* input from joystick       */
    char pos[7];                    /* user's guess at position  */
    unsigned frameno;               /* current video disc frame no. */
    unsigned calcframe();           /* frame number calculator   */
    float d;                        /* distance from home position */
    double sqrt();                  /* square root function      */
```

```
    /* initialize       */
    x = *px; y = *py; z = *pz;
    *dist = 50000;              /* some huge distance          */
    strcpy(pos,"");
    done = FALSE;

    /* find frame and display   */
    player = toggle(player,nplayers);
    frameno = calcframe(x,y,z);
    findframe(player,frameno);
    video(player);
    videoscr();

    /* put up navigation information    */
    drawodom(x,y);
    drawcompass(z);

    /* display prompt, set up window in brackets        */
    clearw(mask,NORMAL);
    for (i = 0; i <= mask[3] - mask[1]; i++)
        c = center(mask,i,prmpt[i]);
    w[0] = c + 1;
    w[1] = w[3] = mask[3];
    w[2] = w[0] + 5;

    /* accept and process input from joystick till button pressed
    while (!done)
        {

        /* save old coords in case new ones are illegal */
        oldx = x; oldy = y; oldz = z;

        /* reset flag indicating joystick movement      */
        sameframe = FALSE;

        /* calculate new coords from joystick input     */
        switch(stat = joystat())
            {
            case FIRE:  /* joystick button pressed      */
                x = *px; y = *py;
                break;
            case FWD:
                travel(&x,&y,z,0);
                break;
            case REV:
                travel(&x,&y,z,8);
                break;
            case RIGHT:
                pivot(&z,15);
                break;
            case LEFT:
                pivot(&z,1);
                break;
            default:
                if (ansgotten = getdigit(w,pos))
                    done = chkpos(x,y,pos,dist,&ntries);
```

```
                sameframe = TRUE;
                break;
            }

        /* if joystick in neutral position, restart loop        */
        if (sameframe)
            continue;

        /* if frame > 30m from home -- sound tone, restart loop */
        if ((d = 12*sqrt((double)(x - *px)*(x - *px) + (y - *py)*(y
0)
            {
            tone(ERRFREQ,ERRTIME);
            x = oldx; y = oldy; z = oldz;
            continue;
            }

        /* if frame is illegal -- sound tone,restart loop       */
        if (getbit(illegal[x][y],z))
            {
            tone(ERRFREQ,ERRTIME);
            x = oldx; y = oldy; z = oldz;
            continue;
            }

        /* search for frame     */
        player = toggle(player,nplayers);
        frameno = calcframe(x,y,z);
        found = findframe(player,frameno);

        /* if error finding frame, restart loop */
        if (!found)
            {
            x = oldx; y = oldy; z = oldz;
            player = toggle(player,nplayers);
            continue;
            }

        /* if joystick position has changed, restart loop       */
        if (joystat() != stat)
             {
             x = oldx; y = oldy; z = oldz;
             player = toggle(player,nplayers);
             continue;
             }

        /* switch the video to the current player       */
        video(player);

        /* update the compass bearing and odometer      */
        compass(z);
        updtodom(x,y);

        } /* end while */

    /* pass values back to calling routine      */
```

```
    *px = x; *py = y; *pz = z;

    /* return # of guesses taken */
    return(ntries);
}
```

```
/*****************************************************************/
/* Name:        atrnav                                           */
/* Description:                                                  */
/*      This program is the driver for the ATR land navigation   */
/*      instruction system running on a modified Hazeltine       */
/*      MicroTICCIT workstation.  The system components are      */
/*      described in the atrdefs.h file.                         */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      This program uses a full screen mode and the async port.*/
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

char window[] = (1,3,78,23);    /* useable portion of screen    */

main()
(
    static char *menutext[] =   /* main menu -- title 0th line  */
        (
        "M A I N    O P T I O N S",
        "Free Travel",
        "Skills Review",
        "Land Navigation Training",
        "Land Navigation Test",
        "Shut Down ATR System"
        );
    static char *routines[] =   /* routines called from menu    */
        (
        "",
        "drive",
        "navrevu",
        "navlssn",
        "navtest",
        "shutdown"
        );
    int nchoices = 5;           /* number of available options  */
    int choice = 1;             /* menu option selected         */

    /* check files, link to serial interface; start players     */
    forkl("startup",NULL);
    if (wait() != NULL)         /* error code from process      */
        return;

    /* display menu of options - execute selected option        */
    while (choice = menu(window,menutext,nchoices,choice))
        (
        clearl(MSGLN,0,80,REVIDEO);
        printf("^[[34;47m   Loading %s module^[[44;37m",menutext[cho
        forkl(routines[choice],NULL);
        wait();
        if (choice == nchoices)         /* shutdown chosen from menu
            return;
        )
)
```

```
/*****************************************************************/
/* Name:        autodrive                                        */
/* Description:                                                  */
/*      This function starts driving at the given coordinates    */
/*      and drives using the given list of instructions until    */
/*      it encounters a HALT instruction.  The instructions are  */
/*      the same as the statuses returned from the steering      */
/*      assembly.                                                */
/* Inputs:      x,y,z - starting position on virtual terrain     */
/*              drvlist - list of driving instructions           */
/* Outputs:     x,y,z - the position after the drive is over     */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

extern char player;             /* currently active player      */
extern char nplayers;           /* number of active players     */

autodrive(px,py,pz,drvlist)
int *px,*py,*pz;
int drvlist[];
{
    int i = 0;                  /* index for instruction list   */
    int try;                    /* number of attempts at frame  */
    int x,y,z;                  /* internal coordinates         */
    int oldx;                   /* previous value of x          */
    unsigned frameno;           /* current video disc frame no. */
    unsigned calcframe();       /* frame number calculator      */

    /* init local x,y,z from parameters */
    oldx = x = *px; y = *py; z = *pz;

    /* find frame on next player and switch video */
    player = toggle(player,nplayers);
    frameno = calcframe(x,y,z);
    for (try = 0; !findframe(player,frameno) && try < 3; try++);
    video(player);
    videoscr();
    drawcompass(z);
    drawodom(x,y);

    do  /* drive until HALT instruction */
        {
        switch(drvlist[i])
            {
            case FWD:
                travel(&x,&y,z,0);
                break;
            case REV:
                travel(&x,&y,z,8);
                break;
            case RIGHT:
                pivot(&z,15);
                break;
            case LEFT:
                pivot(&z,1);
```

```
                break;
            default:
                break;
            )

        /* find correct frame and display       */
        player = toggle(player,nplayers);
        frameno = calcframe(x,y,z);
        for (try = 0; !findframe(player,frameno) && try < 3; try++);

        /* set up delays to smooth out pivots/travels   */
        if (x == oldx)          /* frames < 100 apart   */
            nap(600);

        /* save current x value */
        oldx = x;

        /* switch the video to the current player       */
        video(player);

        /* update the status line on the screen         */
        compass(z);
        updtodom(x,y);

        ) while (drvlist[++i] != HALT);

    /* pass values of x,y,z to calling  routine */
    *px = x; *py = y; *pz = z;
)
```

```
/*****************************************************************/
/* Name:        azimuth                                          */
/* Description:                                                  */
/*      This function takes a student through a short lesson on */
/*      azimuth determination, using a map and protractor, and  */
/*      conversion of that azimuth to a magnetic azimuth, which */
/*      can then be used for navigation.  The student interacts */
/*      with the system by answering several questions.         */
/*      This lesson is part of a larger unit covering land nav- */
/*      gation.                                                 */
/* Inputs:      None                                             */
/* Outputs:     rc - 0 if questions answered correctly           */
/*                 - >0 if questions answered incorrectly       */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

char *adtxt[] =         /* azimuth determination text   */
{
"    ^[[34;47m                    F I N D I N G   A Z I M U T H S
    ^[[44;37m",
"",
"       To figure out the direction you need to travel to go from p
    to point B, you need a protractor, a black marker, and the map.
    ection is measured in degrees and is called an azimuth.",
"",
"       Mark point A at 290790 and point B at 287803 (center of tre
    your map.  Draw a line to connect points A and B.  Find the poi
    where the line crosses a North-South grid line.  Put the protra
    at this point on the map.  Make sure you line up the crossed ce
    lines of the protractor with the grid lines.  Read the azimuth
    rees from the protractor.  The line from A to B will point dire
    the azimuth on the protractor.",
"",
"       What is the grid azimuth, in degrees, from point A to point
"                                    [   ]"
);

char *actxt[] =         /* azimuth conversion text      */
{
"    ^[[34;47m                C O N V E R T I N G   A Z I M U T H S
    ^[[44;37m",
"",
"       The North-South grid lines on the map are different from the
    netic north on your compass.  You will need to convert grid azi
    to magnetic azimuths and magnetic azimuths to grid azimuths.",
"",
"       The difference in degrees between magnetic north and grid n
    the G-M angle.  The G-M angle is pictured in the margin of the
    This is a declination diagram. It also tells you how to change
    azimuths to magnetic azimuths and magnetic azimuths to grid azi
"",
"    What is the ^[[36mmagnetic^[[37m azimuth, in degrees, from poin
 B?",
"                                    [   ]"
```

```
);

extern char window[];           /* useable portion of screen    */

azimuth()
{
    int azmth;                  /* azimuth entered by user      */
    int deg;                    /* difference from actual azmth */
    int ntries = 0;             /* number of guesses made       */
    int rc = 0;                 /* return code to calling pgm   */
    char azmwndw[4];            /* used to get azimuths         */

    /* set up window for response to questions */
    azmwndw[0] = window[0] + 38;
    azmwndw[1] = azmwndw[3] = window[1] + 18;
    azmwndw[2] = window[0] + 40;

    /* azimuth determination review     */
    clearw(window,NORMAL);
    disptext(window,adtxt,3,18);

    /* get answer and evaluate  */
    while (ntries++ < 3)
        {
        azmth = getnum(azmwndw);
        clearl(MSGLN,0,80,REVIDEO);
        if (azmth == 346)
            {
            printf("^[[34;47m   Yes, that's right.^[[44;37m");
            break;
            }
        else if ((deg = abs(azmth - 346)) <= 3)
            {
            printf("^[[34;47m   Good, you're only %d} off.  The corr
 346}.^[[44;37m",deg);
            break;
            }
        else if (ntries == 1)
            printf("^[[34;47m   Sorry, check to see that you lined u
ctor correctly.^[[44;37m");
        else if (ntries == 2)
            printf("^[[34;47m   Sorry, check to see that you marked
 B correctly.^[[44;37m");
        else printf("^[[34;47m   Sorry, the correct anwer is 346}.^[
        }

    if (ntries == 3)
        rc++;

    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    clearl(MSGLN,0,80,REVIDEO);

    /* azimuth conversion review        */
    clearw(window,NORMAL);
```

```
    disptext(window,actxt,3,15);

    /* get user's response to azimuth question */
    azmwndw[1] = azmwndw[3] = window[1] + 15;
    ntries = 0;
    while (ntries++ < 3)
        {
        azmth = getnum(azmwndw);
        clearl(MSGLN,0,80,REVIDEO);
        if (azmth == 338)
            {
            printf("^[[34;47m   Yes, that's right.^[[44;37m");
            break;
            }
        else if ((deg = abs(azmth - 338)) <= 3)
            {
            printf("^[[34;47m   Good, you're only %d° off.  The corr
 346°.^[[44;37m",deg);
            break;
            }
        else if ((deg = abs(azmth - 354)) <= 3)
            {
            if (ntries == 1)
                printf("^[[34;47m   Sorry, you should be converting
to a magnetic azimuth.^[[44;37m");
            else if (ntries == 2)
                printf("^[[34;47m   Sorry, please reread the explana
declination diagram.^[[44;37m");
            }
        else if (ntries < 3)
            printf("^[[34;47m   Sorry, please check the declination
the G-M angle.^[[44;37m");
        else printf("^[[34;47m   No, the magnetic azimuth is 338° (3
muth - 8° G-M angle).^[[44;37m");
        }

    if (ntries == 3)
        rc++;

    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* clear window, message line */
    clearl(MSGLN,0,80,REVIDEO);
    clearw(window,NORMAL);

    /* return whether question answered correctly       */
    return(rc);
}
```

```
/*****************************************************************/
/* Name:        box2                                             */
/* Description:                                                  */
/*      This function draws a double line box around the         */
/*      indicated window.                                        */
/* Inputs:      window - left, top, right, bottom of window      */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
box2(window)
char window[];
{
    register int row,col;

    pcvscp(window[1],window[0]);
    pcvwc(1,201);
    pcvscp(window[1],window[0]+1);
    pcvwc(window[2]-window[0]-1,205);
    pcvscp(window[1],window[2]);
    pcvwc(1,187);
    for(row = window[1]+1; row < window[3]; row++)
        {
        pcvscp(row,window[0]);
        pcvwc(1,186);
        pcvscp(row,window[2]);
        pcvwc(1,186);
        }
    pcvscp(window[3],window[0]);
    pcvwc(1,200);
    pcvscp(window[3],window[0]+1);
    pcvwc(window[2]-window[0]-1,205);
    pcvscp(window[3],window[2]);
    pcvwc(1,188);
    pcvscp(window[3],window[0])
}
```

```
/*****************************************************************/
/* Name:        calcframe                                        */
/* Description:                                                  */
/*      This function calculates a video disc frame number from  */
/*      the three coordinates passed to it. In the process, it   */
/*      corrects for two editing errors in the disc.             */
/*      The (x,y) coordinates each have a minimum value of 2,    */
/*      because values of 0 or 1 are used as a signal that the   */
/*      south and east edges have been reached.  The same two-   */
/*      coordinate pad is used for the maximum x and y values.   */
/* Inputs:      x,y,z - position on virtual terrain              */
/* Outputs:     frameno - frame number corresponding to x,y,z    */
/* Side effects:                                                 */
/*****************************************************************/
#define   STARTFRAME   361

unsigned calcframe(x, y, z)
int x,y,z;
{
    unsigned frameno;

    /* make correction for video disc editing errors    */
    if (((x> 52) && (y > 42)) || (x > 53))
        z = (z - 1) % 16;
    else if (((x > 40) && (y > 16)) || (x > 41))
        z = (z - 2) % 16;

    /* calculate frame number                           */
    frameno = 976 * (x - 2) + 16 * (y - 2) + z + STARTFRAME;
    return(frameno);
}
```

```
/*****************************************************************/
/* Name:        center                                           */
/* Description:                                                  */
/*      This function writes the string provided to the screen,  */
/*      centering it on the given row of the given window.       */
/* Inputs:      window - borders within which to center string   */
/*              row - which line in the window to center string  */
/*              str - character string to be centered            */
/* Outputs:     col - column in which str begins                 */
/* Side effects:                                                 */
/*      cursor is placed off the screen                          */
/*****************************************************************/
center(window,row,str)
char window[],row,*str;
{
    int col;

    curs_off();
    col = 0.5 * (window[0] + window[2] - strlen(str));
    pcvscp(window[1] + row,col);
    printf(str);
    return(col);
}
```

```
/*****************************************************************/
/* Name:        chkpts                                           */
/* Description:                                                  */
/*      This function presents text and videodisc pictures to    */
/*      explain the use of checkpoints in land navigation.       */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

extern char window[];           /* useable portion of screen    */

checkpts()
{

    /* display text */
    clearw(window,NORMAL);
    disptxtf(window,"cptxt",0,3,17);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearw(window,NORMAL);
}
```

```
/*****************************************************************/
/* Name:        chkleg                                           */
/* Description:                                                  */
/*      This function checks to see if the student's position    */
/*      (x,y) is within the limits of the checkpoint location.   */
/*      If not, the student is placed at the checkpoint.         */
/* Inputs:      x,y,z    - student's current position           */
/*              pos      - 6-digit coordinate of checkpoint     */
/*              name     - checkpoint name                      */
/*              limit    - how far student may be from checkpt  */
/* Outputs:     d        - distance from checkpoint             */
/* Side effects:                                                 */
/*      This function must be declared as:                       */
/*              float chkleg();                                  */
/*      It leaves the coordinates displayed on the bottom line   */
/*****************************************************************/
#include "atrdefs.h"

extern char mask[];                 /* masked area of video screen  */
extern char window[];               /* useable portion of screen    */
extern char player;                 /* currently active player      */
extern char nplayers;               /* number of active players     */

float chkleg(px,py,pz,pos,name,limit)
int *px, *py, *pz, limit;
char *pos, *name;
{
    int try;                        /* number of attempts at frame  */
    float d;                        /* distance from checkpoint     */
    float distance();               /* distance calculator          */
    unsigned frameno;               /* current video disc frame no. */
    unsigned calcframe();           /* frame number calculator      */

    /* check distance from checkpoints  */
    clearw(mask,NORMAL);
    videoln(window[3]);
    if ((d = distance(*px,*py,pos)) <= limit)
        (
        cursor(mask[1] + 1,mask[0]);
        curs_off();
        printf("    Good, you are within %d meters of the %s at %s."
pos);
        cursor(mask[1] + 2,mask[0]);
        curs_off();
        printf("    Your current coordinates are shown at the bottom
en.");
        drawcoord(*px,*py);
        )
    else        /* move student to correct position     */
        (
        cursor(mask[1],mask[0]);
        curs_off();
        printf("    Sorry, you are %.0f meters away from the %s at %
os);
        cursor(mask[1] + 1,mask[0]);
        curs_off();
```

```
        printf("    Your current coordinates are shown at the bottom
en.");
        cursor(mask[1] + 3,mask[0]);
        curs_off();
        printf("    Press the button on the joystick and we will pla
.",pos);
        drawcoord(*px,*py);
        joywait(FIRE);
        player = toggle(player,nplayers);
        postoxy(px,py,pos);
        frameno = calcframe(*px,*py,*pz);
        for (try = 0; !findframe(player,frameno) && try < 3; try++);
        delcoord();
        video(player);
        compass(*pz);
        updtodom(*px,*py);
        )
    return(d);
}
```

```
/*****************************************************************/
/* Name:        chkpos                                           */
/* Description:                                                  */
/*      This function checks the user's guesses at his position  */
/*      against his actual position.  He gets up to 3 guesses.   */
/* Inputs:      x,y     - user's actual position                 */
/*              pos     - user's guess at his position           */
/* Outputs:     dist    - how close best guess is to position    */
/*              ntries  - number of guesses taken at position    */
/*              returns TRUE if correct or all guesses taken     */
/*                      FALSE if he gets to try again            */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

chkpos(x,y,pos,dist,ntries)
char *pos;
int x,y,*ntries;
float *dist;
(
    int match;          /* user's answer matched smart answer   */
    float d;            /* distance from exact position - local */
    float distance();   /* routine to calculate distance        */
                        /* messages to user                     */
    static char success[] =
        "^[[34;47m   Good - your answer is within 30 meters of your
on.^[[44;37m";
    static char notfar[] =
        "^[[34;47m   Sorry - your answer is %.0f meters from the cor
44;37m";
    static char failure[] =
        "^[[34;47m   Sorry - that's not correct.^[[44;37m";
    static char rightup[] =
        "^[[34;47m   Remember - READ RIGHT, THEN UP^[[44;37m";
    static char exact[] =
        "^[[34;47m   Your exact coordinates are:^[[44;37m";
    static char retry[] =
        "^[[34;47m   Please try again.^[[44;37m";

    /* increment number of guesses */
    ++(*ntries);

    /* check for correct answer */
    getln(MSGLN,0,80);
    clearl(MSGLN,0,80,REVIDEO);
    if ((d = distance(x,y,pos)) < 30 )
        (
        printf(success);
        nap(3000);
        clearl(MSGLN,0,80,REVIDEO);
        printf(exact);
        dispcoord(x,y);
        *dist = min(*dist,d);
        return(TRUE);
        )
```

```
    else if (distance(-393 - y,-393 - x,pos) < 30) /* reversed coord
        printf("%s%s",failure,rightup);
    else        /* check guess against smart answers */
        printf(notfar,d);

    /* if new guess better than old one, update best guess */
    *dist = min(*dist,d);

    /* see if user has used all his guesses */
    nap(3000);
    clearl(MSGLN,0,80,REVIDEO);
    if (*ntries == 3)
        {
        printf(exact);
        dispcoord(x,y);
        ++(*ntries);    /* ntries > 3 means position not guessed */
        return(TRUE);
        }
    else   /* let user try again */
        {
        printf(retry);
        nap(3000);
        putln(MSGLN,0,80,REVIDEO);
        strcpy(pos,"");
        return(FALSE);
        }
}
```

```
/*****************************************************************/
/* Name:        chktab                                           */
/* Description:                                                  */
/*      This function checks to see if the file of illegal       */
/*      cooordinates can be opened.  It aborts execution with    */
/*      a message if it cannot be opened.                        */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

extern char screen[];                    /* screen definition    */

chktab()
{
    int fid;

    if ((fid = open("badframe.atr",0)) == EOF)
        {
        clearl(MSGLN,0,80,REVIDEO);
        printf("^[[34;47m   Can't open frame table file...press any
program^[[44;37m");
        getch();
        clearw(screen,NORMAL);
        cursor(0,0);
        exit(2);
        }
    close(fid);
}
```

```
/*****************************************************************/
/* Name:        compass,drawcompass,delcompass,readcompass       */
/* Description:                                                  */
/*      These functions provide a digital compass on the screen */
/*      to aid the user in orienting himself on the ATR grid.    */
/* Inputs:      z - current direction user faces                 */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"
#include "stdio.h"
int offset = 8;                          /* declination          */

compass(z)
int z;
{
    static int dir[] =                   /* numeric degrees      */
        {
        360, 342, 315, 297, 270, 252, 225, 207,
        180, 162, 135, 117,  90,  72,  45,  27
        };

    /* display current bearing  */
    cursor(MSGLN,17);
    curs_off();
    printf("^[[47;34m%03d¦^[[44;37m",dir[z] - offset);
}

drawcompass(z)
int z;
{

    /* display compass label    */
    cursor(MSGLN,0);
    curs_off();
    if (offset == 0)
        printf("^[[47;34m  Grid Azimuth: ");
    else
        printf("^[[47;34m  Mag. Azimuth: ");

    /* display current bearing  */
    compass(z);
}

delcompass()
{
    clearl(MSGLN,0,25,REVIDEO);
}
```

```
/*****************************************************************/
/* Name:        drawcoord,dispcoord,delcoord,xytopos,postoxy      */
/* Description:                                                   */
/*      These functions provide a display of the user's           */
/*      position in the terrain.                                  */
/* Inputs:      x,y - coordinates in internal format              */
/* Outputs:     None                                              */
/* Side effects:                                                  */
/*      The internal x,y coordinates are mapped into the 100      */
/*      meter map coordinate system for display purposes.         */
/*****************************************************************/
#include "atrdefs.h"

#define XBASE   2505
#define YBASE   7901

dispcoord(x,y)
int x,y;
{
    char pos[7];                /* 100 meter coordinate string  */

    /* convert x,y coordinates to 100 meter grid coordinate     */
    xytopos(pos,x,y);

    /* display 100 meter grid coordinate        */
    cursor(MSGLN,38);
    curs_off();
    printf("^[[47;34m%s^[[44;37m",pos);
}

drawcoord(x,y)
int x,y;
{

    /* display coord label      */
    cursor(MSGLN,25);
    curs_off();
    printf("^[[47;34m  Position: ");

    /* display current coords   */
    dispcoord(x,y);
}

delcoord()
{
    clearl(MSGLN,25,25,REVIDEO);
}

xytopos(pos,x,y)        /* convert internal x,y to 100m grid    */
char *pos;
int x,y;
{
    x = (XBASE + 12 * (55 - x))/10;
    y = (YBASE + 12 * (y - 2))/10;
    sprintf(pos,"%3d%3d",x,y);
}
```

```
postoxy(px,py,pos)      /* convert 100m grid to internal x,y    */
int *px,*py;
char *pos;
{
    sscanf(pos,"%3d%3d",px,py);
    *px = (660 + XBASE - 10 * (*px))/12;
    *py = (24 - YBASE + 10 * (*py))/12;
}
```

```
/*****************************************************************/
/* Name:        coordsys                                         */
/* Description:                                                  */
/*      This function presents an introduction to the six-digit */
/*      coordinate system used in conjunction with the map for   */
/*      identifying location.  The student is asked to answer    */
/*      an example question as part of the introduction.         */
/* Inputs:      None                                             */
/* Outputs:     rc - 0 if question answered correctly            */
/*                 - 1 if question not answered correctly        */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

char *ctxt[] =          /* coordinate system introduction       */
{
"    ^[[34;47m                    C O O R D I N A T E   S Y S T E M
    ^[[44;37m",
"",
"        Every point on a map is named by a six-digit number, a coor
"    To find a point's six-digit number or name, use the numbered bl
"    lines on the map.  These lines appear every 100 meters and form
"    squares.",
"",
"        Remember to READ RIGHT, THEN UP.  The three digits for East
"    coordinates come before the three digits for North-South coordi
"",
"        Example:",
"            Suppose you are at 240790 (the lower left corner of the
"        If you moved ^[[36mfifty^[[37m meters toward the east, you
24^[[36m5^[[37m790.",
"",
"        Where would you be if you moved fifty meters north of 24079
"                                    [      ]"
};

extern char window[];           /* useable portion of screen    */

coordsys()
{
    int i;                      /* index and loop counter       */
    int ntries = 0;             /* number of guesses made       */
    int rc = 0;                 /* return code to calling pgm   */
    char poswndw[4];            /* used to get coordinates      */
    char str[10];               /* coordinate string            */

    /* set up windows for getpos() to use for coords */
    poswndw[0] = window[0] + 35;
    poswndw[1] = poswndw[3] = window[1] + 18;
    poswndw[2] = window[0] + 40;

    /* display first part of text       */
    clearw(window,NORMAL);
    disptext(window,ctxt,3,11);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
```

```
    clearl(window[3],window[0],window[2] - window[0] + 1,NORMAL);

    /* display second part      */
    disptext(window,ctxt + 9,12,18);

    /* get user's response to coordinate question */
    while (ntries++ < 3)
        (
        getpos(poswndw,str);
        clearl(MSGLN,0,80,REVIDEO);
        if (strcmp("240795",str) == 0)
            (
            printf("^[[34;47m   Yes, that's right.^[[44;37m");
            break;
            )
        else if ((strcmp("245790",str) == 0) || strcmp("235790",str)
            printf("^[[34;47m   Sorry, the first three digits are fo
West coordinate.^[[44;37m");
        else if ((strcmp("240740",str) == 0) || strcmp("240840",str)
            printf("^[[34;47m   Sorry, fifty meters is equal to half
re.^[[44;37m");
        else if (strcmp("240785",str) == 0)
            printf("^[[34;47m   Sorry, remember to read up to get th
h coordinate.^[[44;37m");
        else printf("^[[34;47m   Sorry, please reread the example ca
4;37m");
        nap(3000);
        )

    if (ntries > 3)
        (
        clearl(MSGLN,0,80,REVIDEO);
        printf("^[[34;47m   You would be at 24079^[[31m5^[[37.^[[44;
        rc = 1;
        )
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearl(MSGLN,0,80,REVIDEO);
    clearw(window,NORMAL);

    /* return whether question answered correctly       */
    return(rc);
)
```

```
/*****************************************************************/
/* Name:        curs_on,curs_off,curs_get,curs_put                */
/* Description:                                                   */
/*      These functions use video interrupt 10 to manipulate      */
/*      the cursor.                                               */
/* Inputs:      row,col for curs_put                              */
/* Outputs:     row,col for curs_get                              */
/* Side effects:                                                  */
/*****************************************************************/
#include "dos.h"

union REGS regs;                    /* 8086 registers             */

curs_on()
{

    /* which videc mode?            */
    regs.h.ah = 15;
    int86(0x10,&regs,&regs);

    /* set cursor size              */
    if (regs.h.al == 0x07)          /* monochrome --              */
        {                           /*      9 x 14 pixel grid     */
        regs.h.ch = 12;
        regs.h.cl = 13;
        }
    else                            /*      6 x 8 pixel grid      */
        regs.h.ch = regs.h.cl = 7;

    /* turn on cursor               */
    regs.h.ah = 1;
    int86(0x10,&regs,&regs);
}


curs_off()
{

    /* make cursor invisible        */
    regs.h.ch = regs.h.cl = 15;

    /* turn off cursor              */
    regs.h.ah = 1;
    int86(0x10,&regs,&regs);
}


curs_get(r,c)
char *r,*c;
{

    /* get cursor position          */
    regs.h.ah = 3;
    int86(0x10,&regs,&regs);
    *r = regs.h.dh;
    *c = regs.h.dl;
```

```
}


curs_put(r,c)
char r,c;
{

    /* place cursor at (r,c)     */
    regs.h.ah = 2;
    regs.h.dh = r;
    regs.h.dl = c;
    int86(0x10,&regs,&regs);
}
```

```
/*****************************************************************/
/* Name:        deadreck                                         */
/* Description:                                                  */
/*      This function presents text explaining the concept of    */
/*      dead reckoning as it is used in land navigation.         */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

extern char window[];           /* portion of screen for text   */

deadreck()
{
    struct logentry *log;       /* log of route                 */
    struct logentry *newlog();  /* function returning log       */

    /* display intro text */
    clearw(window,NORMAL);
    clearl(MSGLN,0,80,REVIDEO);
    disptxtf(window,"drtxt",0,3,15);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* show sample of a log     */
    clearw(window,NORMAL);
    disptxtf(window,"drtxt",14,2,4);
    log = newlog();
    drawlog(log);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* show steps to plan a route for dead reckoning    */
    clearw(window,NORMAL);
    disptxtf(window,"drtxt",18,3,17);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* go through step by step example  */
    drckxamp();

    /* let user try one */
    drckprob();

}
```

```
/*****************************************************************/
/* Name:        detazmth                                         */
/* Description:                                                  */
/*      This function takes a student through the process of     */
/*      determining the grid and magnetic azimuths from one      */
/*      point to another.                                        */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

extern char window[];           /* useable portion of screen    */

detazmth()
(
    int azmth;                  /* azimuth entered by user      */
    int deg;                    /* difference from actual azmth */
    int ntries = 0;             /* number of guesses made       */
    int rc = 0;                 /* return code to calling pgm   */
    char azmwndw[4];            /* used to get azimuths         */



    /* azimuth determination text       */
    clearw(window,NORMAL);
    disptxtf(window,"datxt",0,3,10);

    /* set up window for response to question */
    azmwndw[0] = window[0] + 39;
    azmwndw[1] = azmwndw[3] = window[1] + 10;
    azmwndw[2] = window[0] + 41;

    /* get user's response to azimuth question */
    while (ntries++ < 3)
        (
        azmth = getnum(azmwndw);
        clearl(MSGLN,0,80,REVIDEO);
        if (azmth == 150)
            (
            printf("^[[34;47m   Yes, that's right.^[[44;37m");
            break;
            )
        else if ((deg = abs(azmth - 150)) <= 3)
            (
            printf("^[[34;47m   Good, you're only %d} off.  The corr
 150}.^[[44;37m",deg);
            break;
            )
        else if (ntries == 1)
            printf("^[[34;47m   Sorry, check to see that you lined u
ctor correctly.^[[44;37m");
        else if (ntries == 2)
            printf("^[[34;47m   Sorry, check to see that you marked
 B correctly.^[[44;37m");
        else printf("^[[34;47m   Sorry, the correct anwer is 150}.^[
```

```
        )

    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* put correct answer in brackets   */
    clearw(azmwndw,NORMAL);
    cursor(azmwndw[1],azmwndw[0]);
    printf("150");
    clearl(MSGLN,0,80,REVIDEO);

    /* azimuth conversion */
    disptxtf(window,"datxt",8,11,16);
    azmwndw[1] = azmwndw[3] = window[1] + 16;
    ntries = 0;
    while (ntries++ < 3)
        {
        azmth = getnum(azmwndw);
        clearl(MSGLN,0,80,REVIDEO);
        if (azmth == 142)
            {
            printf("^[[34;47m   Yes, that's right.^[[44;37m");
            break;
            }
        else if ((deg = abs(azmth - 142)) <= 3)
            {
            printf("^[[34;47m   Good, you're only %d° off.  The corr
 142°.^[[44;37m",deg);
            break;
            }
        else if ((deg = abs(azmth - 158)) <= 3)
            {
            if (ntries == 1)
                printf("^[[34;47m   Sorry, you should be converting
to a magnetic azimuth.^[[44;37m");
            else if (ntries == 2)
                printf("^[[34;47m   Sorry, please reread the explana
declination diagram.^[[44;37m");
            }
        else if (ntries < 3)
            printf("^[[34;47m   Sorry, please check the declination
the G-M angle.^[[44;37m");
        else printf("^[[34;47m   No, the magnetic azimuth is 142° (1
muth - 8° G-M angle).^[[44;37m");
        }

    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* clear window, message line */
    clearl(MSGLN,0,80,REVIDEO);
    clearw(window,NORMAL);
}
```

```
/*****************************************************************/
/* Name:        disptext                                         */
/* Description:                                                  */
/*      This function displays text in the window passed to it. */
/*      The text is assumed to fit in the window, and is not     */
/*      checked for length.                                      */
/* Inputs:      w - window in which to write text                */
/*              txt - array of pointers to text strings          */
/*              strt - offset from top of window for first line  */
/*              stop - offset from top of window for last line   */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      The cursor is invisible after this routine returns.      */
/*****************************************************************/
#include "atrdefs.h"

disptext(w,txt,strt,stop)
char w[],*txt[];
int strt,stop;
{
    int i;                          /* index and loop counter       */

    curs_off();
    for (i = strt; i <= stop; i++)
        {
        pcvscp(w[1] + i,w[0]);
        printf(*txt++);
        }
}
```

```
/*****************************************************************/
/* Name:        disptxtf                                         */
/* Description:                                                  */
/*      This function displays text from a file in the window.   */
/*      The text is assumed to fit in the window, and is not     */
/*      checked for length.                                      */
/* Inputs:      w - window in which to write text                */
/*              txtfile - name of file containing text           */
/*              offset - first line of file to display           */
/*              strt - offset from top of window for first line  */
/*              stop - offset from top of window for last line   */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      The cursor is invisible after this routine returns.      */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

disptxtf(w,txtfile,offset,strt,stop)
char w[],*txtfile;
int offset,strt,stop;
(
    int i;                          /* index and loop counter       */
    int fid;                        /* file identifier              */
    char c;
    char line[101];                 /* line of text                 */

    /* open file        */
    sprintf(line,"%.8s.atr",txtfile);
    if ((fid = open(line,0)) == EOF)
        (
        clearl(MSGLN,0,80,REVIDEO);
        printf("^[[34;47m   Not able to load text from disk: %s^[[44
        return;
        )

    /* skip past offset */
    for (i = 0; i < offset; i++)
        getline(fid,line,100);

    /* display text     */
    curs_off();
    for (i = strt; i <= stop; i++)
        (
        getline(fid,line,100);
        pcvscp(w[1] + i,w[0]);
        printf(line);
        )

    /* close file       */
    close(fid);
)
```

```
/*****************************************************************/
/* Name:        distance                                         */
/* Description:                                                  */
/*      This function calculates the distance between the two    */
/*      points represented by (lng,lat) and the 6-digit coor-    */
/*      dinate pos.  Both representations are converted to a     */
/*      common 100m scale before the distance calculation.       */
/* Inputs:      lng,lat - internal coordinates (2-55,2-62)       */
/*              pos - 6-digit coordinate pair for map position   */
/* Outputs:     distance between the two points in meters        */
/* Side effects:        pos cannot be less than 6-digits long.   */
/*****************************************************************/
#define XBASE   2500
#define YBASE   7900

float distance(lng,lat,pos)
int lng,lat;
char pos[];
{
    int a,b;
    float x,y;
    double d,sqrt();

    /* convert (lng,lat) to (x,y) on 100m scale */
    x = XBASE + 12*(55 - lng);
    y = YBASE + 12*(lat - 2);

    /* separate 6 digit coordinate into (a,b) on 100m scale     */
    sscanf(pos,"%3d %3d",&a,&b);
    a *= 10;
    b *= 10;

    /* calculate distance from (x,y) to (a,b) */
    d = sqrt((double) (x - a) * (x - a) + (y - b) * ( y - b));

    /* return distance in meters */
    return(d);
)
```

```
/*****************************************************************/
/* Name:        distmeas                                         */
/* Description:                                                  */
/*      This function takes a student throught the process of    */
/*      measuring distance on a map.  Measurement along both     */
/*      straight and curved lines is covered.  Instruction is    */
/*      presented as text and the student must respond to two     */
/*      questions.                                               */
/* Inputs:      None                                             */
/* Outputs:     rc - 0 if both questions answered correctly      */
/*                 - >0 if either question answered incorrectly */
/* Side effects:                                                 */
/*      The text screen is cleared at the end of this function.  */
/*****************************************************************/
#include "atrdefs.h"

char *dmtxt[] =                    /* distance mesurement text     */
{
"    ^[[34;47m                   M E A S U R I N G   D I S T A N C E
    ^[[44;37m",
"",
"        When you travel from one point to another, you need to know
"    distance between the two points.  Use the black marker and stra
"    edge to measure the distance.  Then use the bar scale in the ma
"    of the map to convert that distance to meters.",
"",
"        Mark point C at 260844 and point D at 256830.",
"",
"        What is the distance, in meters, from point C to point D?",
"                                        [     ]"
};

char *cmtxt[] =          /* curved distance mesurement text      */
{
"    ^[[34;47m                   M E A S U R I N G   D I S T A N C E
    ^[[44;37m",
"",
"        You also need to know how to measure the distance between t
"    points around a curve in a road.  Mark point E at 295800 and po
"    at 278810 on your map.",
"",
"        Make a tick mark on your straight edge, and aline the tick
"    with point E.  Aline the paper along the road edge, and make a
"    mark on the straight edge and on the map when you come to a cur
"    Keep doing this until you get to point F.",
"",
"        What is the distance, in meters, from point E to point F?",
"                                        [     ]"
);

extern char window[];              /* useable portion of screen    */

distmeasure()
{
    int dist;                      /* distance entered by user     */
    int ntries = 0;                /* number of guesses made       */
```

```
    int rc = 0;                     /* return code to calling pgm   */
    char distwndw[4];               /* used to get distances        */

    /* set up window for response to questions */
    distwndw[0] = window[0] + 37;
    distwndw[1] = distwndw[3] = window[1] + 13;
    distwndw[2] = window[0] + 41;

    /* distance measurement review      */
    clearw(window,NORMAL);
    disptext(window,dmtxt,3,13);

    /* get user's response to distance question */
    while (ntries++ < 3)
        (
        dist = getnum(distwndw);
        clearl(MSGLN,0,80,REVIDEO);
        if (abs(dist - 147) < 10)
            (
            printf("^[[34;47m   Yes, that's right.^[[44;37m");
            break;
            )
        else if (ntries == 1)
            printf("^[[34;47m   Sorry, check to see that you marked
 D correctly.^[[44;37m");
        else if (ntries == 2)
            printf("^[[34;47m   Sorry, measure the straight line dis
n points C and D.^[[44;37m");
        else printf("^[[34;47m   Sorry, the distance from C to D is
^[[44;37m");
        )

    if (ntries == 3)
        rc++;

    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    clearl(MSGLN,0,80,REVIDEO);

    /* curved distance measurement review       */
    clearw(window,NORMAL);
    disptext(window,cmtxt,3,15);

    /* get user's response to azimuth question */
    distwndw[1] = distwndw[3] = window[1] + 15;
    ntries = 0;
    while (ntries++ < 3)
        (
        dist = getnum(distwndw);
        clearl(MSGLN,0,80,REVIDEO);
        if (abs(dist - 225) <= 10)
            (
            printf("^[[34;47m   Yes, that's right.^[[44;37m");
            break;
            )
```

```
        else if (ntries == 1)
            printf("^[[34;47m   Sorry, check to see that you marked
 F correctly.^[[44;37m");
        else if (ntries == 2)
            printf("^[[34;47m   Sorry, make tick marks more often an
^[[44;37m");
        else printf("^[[34;47m   Sorry, the distance from E to F is
^[[44;37m");
        }

    if (ntries == 3)
        rc++;

    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* clear window, message line */
    clearl(MSGLN,0,80,REVIDEO);
    clearw(window,NORMAL);

    /* return whether question answered correctly       */
    return(rc);
}
```

```
/*****************************************************************/
/* Name:        drckprob                                         */
/* Description:                                                  */
/*      This function takes the user interactively through       */
/*      navigation from one point to another using the tech-     */
/*      nique of dead reckoning.                                 */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

#define X        2
#define Y       61
#define Z        6

int drplst[] =          /* driving instructions for problem     */
    (HALT, HALT);

char *drptxt[] =        /* text for dead reckoning problem      */
{
"    ^[[34;47m              D E A D   R E C K O N I N G   P R A C T I
    ^[[44;37m",
"",
"       We helped you through an example of dead reckoning, so now
"    ready to try one by yourself.  As before, we'll place you at th
"    point and ask you to find your position on the map.  We'll give
"    coordinates of the release point and help determine a route, bu
"    will drive each leg of the route to the release point.  You wil
"    to keep your own log of the route.  After you finish the proble
"    will show you what your completed log should look like.",
"",
"       You are at your start point (314860 on the map).  Mark this p
"   your destination (294835) on the map.  Record the odometer readi
"   log.  We choose not to travel directly to the destination to avo
"   vegetation and trees in our path.  Let's go toward the south fir
"",
"       Set the azimuth on the compass by turning to face south.  Loo
"   map to see how far you should go along this azimuth.  A good poi
"   for is the base of the hill at 314849.  Measure the distance to
"   and after recording it and the azimuth in your log, drive to the
"",
"       Record the odometer reading and the actual distance traveled
"   log.  Looking at the map, you can see that you still need to tra
"   and west to get to the release point.  Let's go west toward the
"   scrub at 294849.  Record the azimuth and distance and drive to t
"",
"       Record the odometer reading and the actual distance traveled
"   log.  You now need to go south to the road at 294835.  Set the a
"   the compass by turning left.  Record the azimuth and the distanc
"   road (from the map) in your log.  Now, drive to the release poin
"",
"       You're at the release point, but you still have to finish log
"   route.  Read the distance traveled off the odometer, calculate t
"   of the last leg, and enter both in the log.  Now, your log is fi
```

```
"",
"    ^[[34;47m              D E A D   R E C K O N I N G   P R A C T I
    ^[[44;37m",
"",
"        Your completed log should look something like this:"
};


extern char window[];           /* useable portion of screen    */
extern char mask[];             /* mask over top part of video  */

drckprob()
{
    int x,y,z;                  /* coordinates on terrain       */
    float navigate();           /* routines returning float     */
    float readodom();
    float chkleg();
    struct logentry *log;       /* log of route taken           */
    struct logentry *newlog();  /* function returning log       */

    /* display explanation of problem   */
    clearw(window,NORMAL);
    disptext(window,drptxt,3,11);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* give user start and release point coordinates    */
    x = X; y = Y; z = Z;
    resetodom(x,y);
    autodriv(&x,&y,&z,drplst);
    log = newlog();
    log->point = 'A';
    log->odom_rdg = readodom();
    clearw(mask,NORMAL);
    disptext(mask,drptxt + 10,0,3);
    center(windo,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* figure out how far to travel along this azimuth; then do it
    log->mag_azmth = 172;
    log->m_dist = 125;
    navigate(&x,&y,&z,drptxt + 15);
    chkleg(&x,&y,&z,"314849","base of the hill",30);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* fill in log for first leg; prep for second and drive it  */
    (log + 1)->point = 'B';
    (log + 1)->odom_rdg = readodom();
    log->a_dist = (log + 1)->odom_rdg - log->odom_rdg;
    (log + 1)->m_dist = 210;
    (log + 1)->mag_azmth = 262;
    navigate(&x,&y,&z,drptxt + 20);
    chkleg(&x,&y,&z,"294849","patch of scrub",30);
```

```
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* finish log for second leg; prep third leg and drive it   */
    (log + 2)->point = 'C';
    (log + 2)->odom_rdg = readodom();
    (log + 1)->a_dist = (log + 2)->odom_rdg - (log + 1)->odom_rdg;
    (log + 2)->m_dist = 140;
    (log + 2)->mag_azmth = 172;
    navigate(&x,&y,&z,drptxt + 25);
    chkleg(&x,&y,&z,"294835","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* finish up log entry for last leg */
    (log + 3)->point = 'D';
    (log + 3)->odom_rdg = readodom();
    (log + 2)->a_dist = (log + 3)->odom_rdg - (log + 2)->odom_rdg;
    clearw(mask,NORMAL);
    disptext(mask,drptxt + 30,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* display completed log    */
    textscr();
    disptext(window,drptxt + 34,2,4);
    drawlog(log);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearw(window,NORMAL);
)
```

```
/*****************************************************************/
/* Name:        drckxamp                                         */
/* Description:                                                  */
/*      This function takes the student through an example of    */
/*      land navigation using the technique of dead reckoning.   */
/*      It consists of traveling several legs of a route under   */
/*      program control, with textual commentary on keeping a    */
/*      log of the route traveled.                               */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

#define X       34              /* coords of starting location  */
#define Y       50
#define Z       13

int drxlst0[] =         /* lists of driving instructions        */
{HALT,HALT};

int drxlst1[] =
{LEFT, LEFT, LEFT, HALT};

int drxlst2[] =
{FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, HALT};

int drxlst3[] =
{RIGHT, RIGHT, RIGHT, RIGHT, HALT};

int drxlst4[] =
{FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FW
 FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FW
};

int drxlst5[] =
{RIGHT, RIGHT, RIGHT, RIGHT, HALT};

int drxlst6[] =
{FWD, FWD, FWD, HALT};

char *drxtxt[] =        /* text for dead reckoning example      */
{
"    ^[[34;47m              D E A D   R E C K O N I N G   E X A M P
    ^[[44;37m",
"",
"        We'll take you through a step-by-step example using dead re
"    to navigate:",
"",
"        1)  Your start point is at 275848.  Your destination is at
"            Mark these two points on your map.",
"",
"        2)  Record the odometer reading as point A in the first col
"            your log",
"",
"    ^[[34;47m              D E A D   R E C K O N I N G   E X A M P
```

```
"",
"       Your log should look like this before the first leg:",
"",
"    We are now at the start point, looking in the directi  of th
"  point.  You can see on the map that there are no good te ain fe
"  tween here and the release point that we can use as checkpoints.
"",
"    Looking at the map, you can see that there are some trees (at
"  and 286855) which will be avoided if we travel to the north.  Le
"  towards the north.",
"",
"    Read the magnetic azimuth off the compass below.  You need to
"  it to a grid azimuth in order to draw your route on the map.  Re
"  azimuths and the declination in your log (columns 4 - 6).",
"",
"    Looking at the map, you can see that there is a road to our n
"  edge of this road (275857) can be our first intermediate point.
"  the distance to the edge of the road and record it in your log.
"  are ready, we'll drive to the road's edge.  Watch the odometer a
,
"",
"    According to the odometer, we've traveled 96 meters on the fi
"  Record the current odometer reading and the actual distance trav
"  the log.  Looking at our new position on the map, you can see th
"  to go east and south to reach our release point.  Let's go east
"",
"    Record the magnetic and grid azimuths in your log.  Draw a li
,
"  present location due east.  The point on the line which is due n
,
"  destination is our next intermediate point, point C.  Record the
"  distance from here to point C.  Watch the odometer as we drive t
"",
"    We've arrived at intermediate point C.  Record the odometer r
"  column 1 of your log.  Subtract the previous odometer reading to
"  actual distance traveled on this leg and record it in column 3.
"  lease point is directly south, so let's turn south.",
"",
"    Record the magnetic and grid azimuths in columns 4 - 6.  Then
"  the distance from our present position to the release point and
"  in column 2.  When you're ready, we'll drive to the release poin
"  ber to watch the odometer.",
"",
"    We're here at the release point at 311844.  Record the odomet
"  in column 1 of your log and fill out the rest of the line for th
,
"",
"    ^[[34;47m              D E A D   R E C K O N I N G   E X A M P
    ^[[44;37m",
"",
"       Your completed log should something look like this:"
};

extern char player;              /* currently active player      */
extern char mask[];              /* masked area of video screen  */
extern char window[];            /* useable portion of screen    */
```

```
drckxamp()
{
    int x,y,z;                      /* coordinates on terrain       */
    float readodom(),navigate();/* routines returning float     */
    struct logentry *log;           /* log of route taken           */
    struct logentry *newlog();  /* funtion returning log        */

    /* introduction to example  */
    x = X; y = Y; z = Z;
    resetodom(x,y);
    log = newlog();
    log->point = 'A';
    log->odom_rdg = readodom();
    clearw(window,NORMAL);
    disptext(window,drxtxt,3,12);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* display partially filled in log  */
    clearw(window,NORMAL);
    disptext(window,drxtxt + 11,2,4);
    drawlog(log);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* show terrain at start point facing release point */
    autodriv(&x,&y,&z,drxlst0);
    clearw(mask,NORMAL);
    disptext(mask,drxtxt + 15,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    videoln(window[3]);

    /* decide to go north       */
    clearw(mask,NORMAL);
    disptext(mask,drxtxt + 19,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,drxlst1);

    /* figure out and record azimuths   */
    clearw(mask,NORMAL);
    log->mag_azmth = 352;
    disptext(mask,drxtxt + 23,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    videoln(window[3]);

    /* measure distance to checkpoint and drive to it   */
    clearw(mask,NORMAL);
    log->m_dist = 80;
    disptext(mask,drxtxt + 27,0,3);
```

```
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,drxlst2);

    /* stop at road edge; turn east     */
    clearw(mask,NORMAL);
    (log + 1)->point = 'B';
    (log + 1)->odom_rdg = readodom();
    log->a_dist = (log + 1)->odom_rdg - log->odom_rdg;
    disptext(mask,drxtxt + 32,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,drxlst3);

    /* record azimuths and distance to next checkpoint; drive to it
    clearw(mask,NORMAL);
    (log + 1)->m_dist = 367;
    (log + 1)->mag_azmth = 82;
    disptext(mask,drxtxt + 37,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,drxlst4);

    /* arrive at second checkpoint; turn to face release point  */
    clearw(mask,NORMAL);
    (log + 2)->point = 'C';
    (log + 2)->odom_rdg = readodom();
    (log + 1)->a_dist = (log + 2)->odom_rdg - (log + 1)->odom_rdg;
    disptext(mask,drxtxt + 42,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,drxlst5);

    /* record azimuths and distance to release point; drive to it
    clearw(mask,NORMAL);
    (log + 2)->m_dist = 65;
    (log + 2)->mag_azmth = 172;
    disptext(mask,drxtxt + 47,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,drxlst6);

    /* record odometer; finish filling out log  */
    clearw(mask,NORMAL);
    (log + 3)->point = 'D';
    (log + 3)->odom_rdg = readodom();
    (log + 2)->a_dist = (log + 3)->odom_rdg - (log + 2)->odom_rdg;
    disptext(mask,drxtxt + 52,0,2);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
```

```
    /* show completed log */
    textscr();
    disptext(window,drxtxt + 55,2,4);
    drawlog(log);
    center(window,window[3] - window[1],"Press button on joystick tc
    joywait(FIRE);
    clearw(window,NORMAL);
)
```

```
/*****************************************************************/
/* Name:        drive                                            */
/* Description:                                                  */
/*      This function takes input from the steering assembly     */
/*      and translates it first into coordinates and then into   */
/*      frame numbers for the video disc player.  It checks      */
/*      for illegal action by the driver, signals them by a      */
/*      tone from the computer, and ignores those actions.       */
/* Inputs:      x,y,z - starting position on virtual terrain     */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

char *pg[] =                        /* introductory text         */
{
"    ^[[34;47m                              F R E E   T R A V E L
   .^[[44;37m",
"",
"        This segment is a demonstration of ATR's capability to simu
"    travel.",
"",
"        You may travel over the simulated terrain using the joystic
"    control your movement.  Moving the joystick forward or backward
"    move you in that direction.  Moving the joystick to the right o
"    will only turn you toward that direction; you will not move for
"    or backward.",
"",
"    Note:  Sometimes, you may attempt to travel over terrain that c
"           be traversed using this system; if this happens, the com
"           will sound a tone, and you will stay where you are."
);

/* global variables     */
int illegal[58][65];                /* bit map of illegal frames    */
char player = 0;                    /* currently active player      */
char nplayers = 1;                  /* number of active players     */
char screen[] = {0,0,79,24};        /* screen definition            */
char window[] = {1,3,78,23};        /* portion of screen for text   */

main()
{
    unsigned frameno;               /* current video disc frame no. */
    unsigned calcframe();           /* declare calcframe function   */
    int x,y,z;                      /* coords of current position   */
    int oldx,oldy,oldz;             /* coords of previous position  */
    int sameframe;                  /* flag to show no change       */
    int found;                      /* frame found on disc?         */
    int stat;                       /* input from joystick          */
    int i;                          /* index and loop counter       */

    /* load illegal frame table */
    loadtab();

    /* introductory text */
    clearw(window,NORMAL);
```

```
    clearl(MSGLN,0,80,REVIDEO);
    disptext(window,pg,3,16);
    center(window,window[3] - window[1]," Press button on joystick t
    joywait(FIRE);

    /* start somewhere in terrain       */
    randloc(&x,&y,&z);

    /* find frame on next player and switch video */
    player = toggle(player,nplayers);
    frameno = calcframe(x,y,z);
    findframe(player,frameno);
    video(player);
    videoscr();

    /* display compass, odometer and coordinates        */
    drawcompass(z);
    drawcoord(x,y);
    resetodom(x,y);
    drawodom(x,y);
    center(window,window[3] - window[1]," Press button on joystick t

    while (TRUE)        /* let user drive till button hit */
        (

        /* save old coords in case new ones are illegal */
        oldx = x; oldy = y; oldz = z;

        /* reset flag indicating joystick movement      */
        sameframe = FALSE;

        /* calculate new coords from joystick input     */
        switch(stat = joystat())
            {
            case FIRE:  /* joystick button pressed      */
                delcoord();
                delcompass();
                delodom();
                textscr();
                return;
                break;
            case FWD:
                travel(&x,&y,z,0);
                break;
            case REV:
                travel(&x,&y,z,8);
                break;
            case RIGHT:
                pivot(&z,15);
                break;
            case LEFT:
                pivot(&z,1);
                break;
            default:
                sameframe = TRUE;
                break;
```

```
        )

    /* if joystick in neutral position, restart loop        */
    if (sameframe)
        continue;

    /* if frame is illegal -- sound tone, restart loop      */
    if (getbit(illegal[x][y],z))
        (
        tone(ERRFREQ,ERRTIME);
        x = oldx; y = oldy; z = oldz;
        continue;
        )

    /* search for frame     */
    player = toggle(player,nplayers);
    frameno = calcframe(x,y,z);
    found = findframe(player,frameno);

    /* if error finding frame, restart loop */
    if (!found)
        (
        x = oldx; y = oldy; z = oldz;
        player = toggle(player,nplayers);
        continue;
        )

    /* if joystick position has changed, restart loop       */
    if (joystat() != stat)
         (
         x = oldx; y = oldy; z = oldz;
         player = toggle(player,nplayers);
         continue;
         )

    /* set up delays to smooth out travels  */
    switch(stat)
        (
        case FWD:
        case REV:
            if (x == oldx)  /* frames < 100 apart   */
                nap(400);
            break;
        default:
            break;
        )

    /* switch the video to the current player       */
    video(player);

    /* update the status line on the screen         */
    compass(z);
    dispcoord(x,y);
    updtodom(x,y);

    ) /* end of drive loop  */
```

```
}
```

```
/*****************************************************************/
/* Name:        findframe                                        */
/* Description:                                                  */
/*      This function causes the LDP-1000A to search for the     */
/*      given frame number. It waits until the frame is found,   */
/*      as indicated by a return code from the player.           */
/* Inputs:      player - which player to use                     */
/*              frameno - which frame to search for              */
/* Outputs:     TRUE - if frame was successfully found           */
/*              FALSE - if error occurred                        */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"
#define RETRY   3

char vddigits[] =
    {0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39};

findframe(player, frameno)
char player;
unsigned frameno;
{
    int sc;                     /* status code from player      */
    int ntries = 0;             /* number of retries            */

    /* send command sequence to find frame */
    while (ntries++ < RETRY)
        {
        /* send search command  */
        if ((sc = vdcmd(player,SEARCH)) != ACK) /* search not accept
            {
            sc = vdcmd(player,CLEAR);
            continue;
            }

        /* send frame number    */
        sendfnum(player,frameno);

        /* send enter command   */
        if ((sc = vdcmd(player,ENTER)) == ACK)
            break;

        sc = vdcmd(player,CLEAR);
        }

    if (ntries >= RETRY)        /* not able to complete sequence
        return(FALSE);

    while (((sc = pcarc(0)) >> 8) & 0x80)/* TICCIT modification
        ;                               /* ignore time-out status

    if (sc == DONE)     /* frame found  */
        return(TRUE);
    return(FALSE);
}
```

```
/*****************************************************************/
/* Name:        getbit                                           */
/* Description:                                                  */
/*      This function gets the value of the bit in position      */
/*      pos from number and returns it as an integer.            */
/* Inputs:      pos - position of bit in number (0 - 15)         */
/*              number - 16 bit integer                          */
/* Outputs:     whether bit in position pos is lit or not        */
/* Side effects:                                                 */
/*****************************************************************/
int getbit(number, pos)
int number, pos;
{
    return((number >> pos) & 01);
}
```

```
/*****************************************************************/
/* Name:        getdigit()                                       */
/* Description:                                                  */
/*      This function accepts one key from the keypad each       */
/*      time it is called.  It returns a FALSE value until 6     */
/*      digits have been entered and the RETURN key pressed, at  */
/*      which point it returns a TRUE value.  Only digits 0-9,   */
/*      carriage return and backspace are allowed as input.      */
/* Inputs:      w - field where 6 digit number is to be entered  */
/*              str - variable which will contain the number     */
/* Outputs:     str - contains the number entered by the user    */
/* Side effects:                                                 */
/*      If the str parameter is empty (i.e. has length 0), the   */
/*      function reinitializes.                                  */
/*      Error messages are displayed on MSGLN as defined in the  */
/*      atrdefs.h file.                                          */
/*****************************************************************/
#include "atrdefs.h"
#include "ctype.h"

extern char window[];           /* useable portion of screen    */

getdigit(w,str)
char w[],str[];
{
    int c;                      /* character entered from keypad*/
    static int ndigits;         /* number of digits entered     */
    static char col;            /* position of cursor in window */
    static char ncols;          /* number of columns in window  */

    /* initialize variables, blank input window */
    if (strlen(str) == 0)
        {
        strcpy(str," ");
        ndigits = 0;
        col = w[0];
        ncols = w[2] - w[0] + 1;
        clearw(w,NORMAL);
        center(window,window[3] - window[1]," Press '*' to backspace
nter answer ");
        }

    /* process keypad key if one has been hit */
    cursor(w[1],col);
    if (isdigit(c = joystat()))
        {
        tone(1000,1);
        str[ndigits] = c;
        putch(c);
        if (col < w[2])
            col++;
        if (ndigits < ncols)    /* not in last column   */
            ndigits++;
        cursor(w[1],col);
        keywait(c);             /* wait till keypad character releas
        }
```

```
    else if ((c == '\b') && (col > w[0]))       /* backspace charact
        {
        tone(500,1);
        ndigits--;
        if (ndigits < ncols - 1)
            col--;
        cursor(w[1],col);
        putch(' ');
        cursor(w[1],col);
        keywait(c);             /* wait till keypad character releas
        }
    else if (c == '\n') /* carriage return      */
        {
        tone(750,1);
        keywait(c);             /* wait till keypad character releas
        if (ndigits == ncols)
            {

            /* terminate string, clear "Press..." message and return
            str[ndigits] = '\0';
            videoln(window[3]);
            return(TRUE);
            }

        /* present error message, restore intructions   */
        getln(MSGLN,0,80);
        clearl(MSGLN,0,80,REVIDEO);
        printf("^[[34;47m Please enter a 6 digit coordinate^[[44;37m
        nap(3000);
        putln(MSGLN,0,80);
        cursor(w[1],col);
        }

    /* user has not entered all digits */
    return(FALSE);
}
```

```
/*****************************************************************/
/* Name:        getline                                          */
/* Description:                                                  */
/*      This function reads a line from a file and puts it in    */
/*      the buffer passed to it.                                 */
/* Inputs:      fid - file to read from                          */
/*              buf - buffer to read line into                   */
/*              lim - length of buffer                           */
/* Outputs:     i   - number of characters read                  */
/* Side effects:                                                 */
/*      \n character is stripped from line                       */
/*****************************************************************/
#include "stdio.h"

getline(fid,buf,lim)
int fid,lim;
char buf[];
{
    int c = 0;                  /* character read               */
    int i = 0;                  /* number of characters read    */

    while ((i < lim) && (c != '\n'))
        {
        read(fid,&c,1);
        if (c != '\n')
            buf[i++] = c;
        }

    /* terminate string */
    buf[i] = '\0';
    return(i);
}
```

```
/*****************************************************************/
/* Name:        getname                                          */
/* Description:                                                  */
/*      This function gets a name from the TICCIT keyboard.      */
/* Inputs:      name -  user's name                              */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      Name limited to 25 characters                            */
/*****************************************************************/
#include "atrdefs.h"
#include "ctype.h"

extern char window[];           /* useable portion of screen    */

getname(name)
char name[];
{
    int c;                      /* keyboard character entered   */
    int len;                    /* number of chars in password  */
    int nchars;                 /* number of chars entered      */
    int ntries;                 /* number of guesses allowed    */
    char row,col;               /* position of cursor in window */

    /* initialize variables, blank input window */
    pcvgcp(&row,&col);
    curs_off();
    nchars = ntries = 0;
    len = 25;
    pcvwca(len,' ',REVIDEO);

    /* process keys till user hits carriage return */
    while (c = getch())
        {
        clearl(MSGLN,0,80,REVIDEO);
        if (nchars < len && isprint(c))
            {
            pcvscp(row,col + nchars);
            pcvwca(1,c,NORMAL);
            name[nchars++] = c;
            }
        else if (c == '\r')             /* carriage return      */
            {
            if (nchars == 0)
                name[nchars++] = ' ';
            name[nchars] = '\0';
            pcvscp(row,col + nchars);
            pcvwca(len - nchars,' ',NORMAL);
            return;
            }
        else if (c == '\b' && nchars > 0)       /* backspace charact
            {
            pcvscp(row,col + --nchars);
            pcvwca(1,' ',REVIDEO);
            }
        } /* end while */
}
```

```
/*****************************************************************/
/* Name:        getnum                                           */
/* Description:                                                  */
/*      This function gets a number from the user as it is       */
/*      entered from the keypad on the joystick.                 */
/*      Only digits 0-9, carriage return ('#' on the keypad),    */
/*      and backspace ('*' on the keypad) are allowed as input.  */
/* Inputs:      w - field where 6 digit number is to be entered  */
/*              str - variable which will contain the number     */
/* Outputs:     str - contains the number entered by the user    */
/* Side effects:                                                 */
/*      Error messages are displayed on MSGLN as defined in the  */
/*      atrdefs.h file.                                          */
/*****************************************************************/
#include "atrdefs.h"
#include "ctype.h"

extern char window[];           /* useable portion of screen    */

getnum(w)
char w[];
{
    int num;                    /* user's repsonse              */
    int c;                      /* character entered from keypad*/
    int ncols;                  /* number of columns in window  */
    int ndigits;                /* number of digits entered     */
    char col;                   /* position of cursor in window */

    /* initialize variables, blank input window */
    ndigits = num = 0;
    col = w[0];
    ncols = w[2] - w[0] + 1;
    clearw(w,NORMAL);
    center(window,window[3] - window[1],"Press '*' to backspace or '
answer");
    cursor(w[1],col);

    /* process keys till user hits carriage return */
    while (TRUE)
        {
        if (isdigit(c = joystat()))
            {
            tone(1000,1);
            num = 10 * num + (c - '0');
            putch(c);
            if (col < w[2])
                col++;
            if (ndigits < ncols)        /* not in last column   */
                ndigits++;
            }
        else if ((c == '\b') && (col > w[0]))   /* backspace charact
            {
            tone(500,1);
            num /= 10;
            if (ndigits-- < ncols)      /* don't move cursor if last
                col--;
```

```
            cursor(w[1],col);
            putch(' ');
            cursor(w[1],col);
            }
        else if ((c == '\n') && (col > w[0]))   /* carriage return
            {
            tone(750,1);
            keywait(c);
            clear1(window[3],window[0],window[2] - window[0] + 1,NOR
            return(num);
            }

        /* reposition cursor in window  */
        cursor(w[1],col);
        keywait(c);             /* wait till keypad character releas
        } /* end while */
}
```

```
/*****************************************************************/
/* Name:        getpos                                           */
/* Description:                                                  */
/*      This function gets the user's position on the terrain    */
/*      board as it is entered from the keypad on the joystick.  */
/*      Only digits 0-9, carriage return ('#' on the keypad),    */
/*      and backspace ('*' on the keypad) are allowed as input.  */
/* Inputs:      w - field where 6 digit number is to be entered  */
/*              str - variable which will contain the number     */
/* Outputs:     str - contains the number entered by the user    */
/* Side effects:                                                 */
/*      Error messages are displayed on MSGLN as defined in the  */
/*      atrdefs.h file.                                          */
/*****************************************************************/
#include "atrdefs.h"
#include "ctype.h"

extern char window[];           /* useable portion of screen    */

getpos(w,str)
char w[],str[];
{
    int c;                      /* character entered from keypad*/
    int ncols;                  /* number of columns in window  */
    int ndigits;                /* number of digits entered     */
    char col;                   /* position of cursor in window */

    /* initialize variables, blank input window */
    ndigits = 0;
    col = w[0];
    ncols = w[2] - w[0] + 1;
    clearw(w,NORMAL);
    center(window,window[3] - window[1],"Press '*' to backspace or '
answer");
    cursor(w[1],col);

    /* process keys till user hits carriage return */
    while (TRUE)
        {
        if (isdigit(c = joystat()))
            {
            tone(1000,1);
            str[ndigits] = c;
            putch(c);
            if (col < w[2])
                col++;
            if (ndigits < ncols)        /* not in last column   */
                ndigits++;
            }
        else if ((c == '\b') && (col > w[0]))   /* backspace charact
            {
            tone(500,1);
            if (ndigits-- < ncols)      /* don't move cursor if last
                col--;
            cursor(w[1],col);
            putch(' ');
```

```
            cursor(w[1],col);
            }
        else if (c == '\n')     /* carriage return      */
            {
            tone(750,1);
            keywait(c);
            if (ndigits == ncols)
                {
                /* terminate string, clear message line and return
                str[ndigits] = '\0';
                clearl(window[3],window[0],window[2] - window[0] + 1
                return;
                }

            /* present error message    */
            getln(MSGLN,0,80);
            clearl(MSGLN,0,80,REVIDEO);
            printf("^[[34;47m Please enter a 6 digit coordinate^[[44
            nap(3000);
            putln(MSGLN,0,80);
            }

        /* reposition cursor in window  */
        cursor(w[1],col);
        keywait(c);

        } /* end while */
}
```

```
/*****************************************************************/
/* Name:        getpw                                            */
/* Description:                                                  */
/*      This function gets a password from the TICCIT keyboard.  */
/*      The user is given three tries to correctly enter the     */
/*      password.                                                */
/* Inputs:      passwd - password to compare user's entry to     */
/* Outputs:     TRUE - if password entered correctly             */
/*              FALSE - if password not entered correctly        */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"
#include "ctype.h"

extern char window[];           /* useable portion of screen    */

getpw(passwd)
char passwd[];
{
    int c;                      /* keyboard character entered   */
    int len;                    /* number of chars in password  */
    int nchars;                 /* number of chars entered      */
    int ntries;                 /* number of guesses allowed    */
    char row,col;               /* position of cursor in window */
    char str[10];               /* string to hold user's entry  */

    /* initialize variables, blank input window */
    pcvgcp(&row,&col);
    curs_off();
    nchars = ntries = 0;
    len = strlen(passwd);
    pcvwca(len,' ',REVIDEO);

    /* process keys till user hits carriage return */
    while (TRUE)
        {
        c = getch();
        clearl(MSGLN,0,80,REVIDEO);
        if (isalpha(c) && nchars < len)
            {
            pcvscp(row,col + nchars);
            pcvwca(1,' ',NORMAL);
            str[nchars++] = toupper(c);
            }
        else if (c == '\b' && nchars > 0)       /* backspace charact
            {
            pcvscp(row,col + --nchars);
            pcvwca(1,' ',REVIDEO);
            }
        else if (c == '\r')     /* carriage return      */
            {
            str[nchars] = '\0';
            if (strcmp(passwd,str) == 0)        /* correct      */
                return(TRUE);
            else if (ntries++ < 2)              /* try again    */
                {
```

```
                printf("^[[34;47m   Sorry, please try again^[[44;37m
                nchars = 0;
                pcvscp(row,col);
                pcvwca(len,' ',REVIDEO);
                }
            else                            /* no more chances      */
                return(FALSE);
            }
        ) /* end while */
}
```

```
/*****************************************************************/
/* Name:        gettry                                           */
/* Description:                                                  */
/*      This function gets a number the TICCIT keyboard.  Only   */
/*      1,2 or 3 are accepted.                                   */
/* Inputs:      None                                             */
/* Outputs:     num - number entered                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

gettry()
{
    int c;                          /* keyboard character entered   */
    int num = 0;                    /* number entered by user       */
    char row,col;                   /* position of cursor in window */

    /* initialize variables, blank input window */
    pcvgcp(&row,&col);
    curs_off();
    pcvwca(1,' ',REVIDEO);

    /* process keys till user hits carriage return */
    while (1)
        {
        if ('1' <= (c = getch()) && c <= '3')
            {
            num = c - '0';
            pcvwca(1,c,NORMAL);
            }
        else if (c == '\b')     /* backspace character  */
            pcvwca(1,' ',REVIDEO);
        else if (c == '\r' && num > 0)          /* carriage return
            return(num);
        } /* end while */
}
```

```
hline(row,col,n)        /* draws horizontal line with -| ends   */
char row,col,n;
{
    pcvscp(row,col);
    pcvwc(1,'Ú');
    pcvscp(row,col+1);
    pcvwc(n - 1,'Ä');
    pcvscp(row,col+n);
    pcvwc(1,'Ñ');
}
```

```
/*****************************************************************/
/* Name:        initlink                                         */
/* Description:                                                  */
/*      This function initializes the link through the serial    */
/*      port to the switching device.  If power to the switch    */
/*      is not on, a message is displayed until power is turned  */
/*      on. The serial buffer is cleared and the switch reset.   */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

initlink()
{

    /* set up serial port mode -- bps, parity, data/stop bits   */
    pcasm(0,MODE);

    /* clear serial port buffer */
    while (pcags(0) & 0x0100)
        pcarc(0);

    /* check interface unit for power   */
    while (TRUE)
        {
        select(JOYSTICK);
        nap(1000);

        /* if buffer contains joystick statuses, power is on    */
        if ((pcags(0) & 0x0100) && ((pcarc(0) & 0xff) >= 0x80))
            break;

        /* interface unit not powered up -- display message     */
        cursor(MSGLN,1);
        curs_off();
        printf("^[[34;47m  Please switch on power to the interface u
");
        }

    /* reset multiplexor */
    select(NULL);

    /* clear serial port buffer */
    while (pcags(0) & 0x0100)
        pcarc(0);

    /* clear message line       */
    clearl(MSGLN,0,80,REVIDEO);
}
```

```
/*****************************************************************/
/* Name:        joystat                                          */
/* Description:                                                  */
/*      This function reads and decodes the joystick status,     */
/*      then sets return codes depending on that status.         */
/* Inputs:      None                                             */
/* Outputs:     4 bit direction - if joystick moved              */
/*              ASCII code      - if keypad pressed              */
/*              -1              - if fire button pressed         */
/*              0               - default                        */
/* Side effects:                                                 */
/*      This function assumes that the joystick is selected.     */
/*****************************************************************/
#include "atrdefs.h"

char kpdigits[] =                   /* digits from joystick keypad  */
    {'\0','1','2','7','6','3','\n','\0','9','\b','5','8','0','\0','4

joystat()
{
    int sc;                         /* status code from joystick    */

    /* select joystick  */
    select(JOYSTICK);

    /* set return code based on joystick status read    */
    sc = pcarc(0) & 0xff;
    if ((sc >= 0x80) && (sc <= 0x8e))           /* keypad press
        sc = kpdigits[sc & 0x0f];
    else if ((sc >= 0xa1) && (sc <= 0xaa))      /* joystick movement
        sc &= 0xff;
    else if ((sc == 0xa0) || ((sc >= 0xb0) && (sc <= 0xb8)))
        sc = -1;                                /* fire button press
    else                                        /* no action
        sc = 0;

    /* return status from joystick      */
    return(sc);
}
```

```
/*************************************************************************/
/* Name:        getln(),putln()                                          */
/* Description:                                                          */
/*      These function read/write and buffer one line from               */
/*      the screen.                                                      */
/* Inputs:      row,col - starting coordinates for read/write            */
/*              n - number of characters to buffer                       */
/* Outputs:     None                                                     */
/* Side effects:                                                         */
/*      cursor positioned off the screen                                 */
/*      attributes left unchanged                                        */
/*************************************************************************/

char ln[80];            /* line buffer  */

getln(row,col,n)
char row,col;
int n;
{
    register int i;
    register char a,c;
    register short ac;

    curs_off();
    for (i = 0; i < n; i++)
        {
        pcvscp(row,col + i);
        ln[i] = pcvrca() & 0xff;
        }
}

putln(row,col,n)
char row,col;
int n;
{
    register int i;
    register char a,c;
    register short ac;

    curs_off();
    for (i = 0; i < n; i++)
        {
        pcvscp(row,col + i);
        pcvwc(1,ln[i]);
        }
}
```

```
/*****************************************************************/
/* Name:        loadtab                                          */
/* Description:                                                  */
/*      This function opens the file of illegal coordinates      */
/*      for the ATR grid and loads a reference table.  The       */
/*      table contains an entry for each grid center which is    */
/*      16 bits deep, one for each direction. A 0 value means    */
/*      that display of that frame is allowed; a 1 value means   */
/*      that the frame cannot be displayed, usually because of   */
/*      editting errors in the disk or physical obstructions.    */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      This routine assumes the file exists.  This is checked   */
/*      for in startup routine.                                  */
/*****************************************************************/

extern int illegal[58][65];              /* illegal frame table  */

loadtab()
{
    int fid;

    /* open file */
    fid = open("badframe.atr",0);

    /* load table from disk file        */
    read(fid,illegal,2*58*65);
    close(fid);
}
```

```
/*****************************************************************/
/* Name:        menu                                             */
/* Description:                                                  */
/*      This function presents an unnumbered menu of choices.    */
/*      The current choice is highlighted and can be changed     */
/*      using some input device (keyboard, joystick, mouse).     */
/*      Modify the #defines and the entermsg variable to match   */
/*      the input device.                                        */
/* Inputs:      w - portion of screen available for menu         */
/*              body - list of menu choices (0th is title)       */
/*              npicks - number of available choices             */
/*              curpick - last choice made                       */
/* Outputs:     curpick - new choice                             */
/* Side effects:                                                 */
/*      Strings longer than the window is wide are truncated.    */
/*      Assumes bottom line available for messages.              */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

#define DOWN    0xa8
#define UP      0xa4
#define ENTER   FIRE
#define getinput()      joystat()
#define clrinput()      select(JOYSTICK); while (joystat() == FIRE)

char entermsg[] =               /* menu prompt
"^[[34;47m  Use the joystick to highlight your choice, then press th
on^[[44;37m";

menu(w,body,npicks,curpick)
char w[],*body[];
int npicks,curpick;
{
    char mw[4];                 /* window for centered menu display
    int i;                      /* index and loop counter
    int width,height;           /* dimensions of menu window
    int oldpick;                /* previous menu choice

    /* width of window is longest string -- plus spaces and borders
    width = strlen(body[0]);
    for (i = 1; i <= npicks; i++)
        width = max(width,strlen(body[i]));
    width += 4;

    /* height is number of choices plus title and borders       */
    height = npicks + 3;

    /* set up menu window -- centered horizontally and vertically
    mw[0] = 0.5 * (w[0] + w[2] - width);
    mw[2] = mw[0] + width;
    mw[1] = 0.5 * (w[1] + w[3] - (4 + npicks));
    mw[3] = mw[1] + height;

    /* draw box around window, display menu title and choices   */
    clearl(MSGLN,0,80,REVIDEO);
```

```
    clearw(mw,NORMAL);
    box2(mw);
    hline(mw[1] + 2,mw[0],width);
    center(mw,1,body[0]);
    for (i = 1; i <= npicks; i++)
        {
        cursor(mw[1] + 2 + i,mw[0] + 2);        /* left justify */
        printf(body[i]);
        }

    /* display instructions, mark previous choice       */
    cursor(MSGLN,1);
    curs_off();
    printf(entermsg);
    revideo(mw[1] + 2 + curpick,mw[0] + 1,width - 1);

    /* process input    */
    clrinput();
    while (TRUE)
        {
        oldpick = curpick;
        switch(getinput())
            {
            case DOWN:
                if (++curpick > npicks)
                    curpick = 1;        /* wrap to top  */
                break;
            case UP:
                if (--curpick < 1)
                    curpick = 1;        /* don't wrap   */
                break;
            case ENTER: /* remove menu, clear msg line  */
                clearl(MSGLN,0,80,REVIDEO);
                clearw(mw,NORMAL);
                return(curpick);
                break;
            default:
                break;
            }

        /* unmark previous choice, mark current one     */
        if (curpick != oldpick)
            {
            normal(mw[1] + 2 + oldpick,mw[0] + 1,width - 1);
            revideo(mw[1] + 2 + curpick,mw[0] + 1,width - 1);
            nap(200);
            }
    }
}
```

```
/*****************************************************************/
/* Name:        nap                                              */
/* Description:                                                  */
/*      This routine waits for approximately the number of       */
/*      milliseconds passed to it. It is completely dependent    */
/*      on the clock rate of the computer.                       */
/* Inputs:      msecs - the number of milliseconds to wait       */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
nap(msecs)
int msecs;
{
    int i,j;

    for (i = 0; i < msecs; i++)
        for (j = 0; j < 50; j++);  /* 50 loops = 1/1000th? */
)
```

```
/*****************************************************************/
/* Name:        navigate                                         */
/* Description:                                                  */
/*      This function takes input from the steering assembly     */
/*      and translates it first into coordinates and then into   */
/*      frame numbers for the video disc player.  It checks      */
/*      for illegal action by the driver, signals them by a      */
/*      tone from the computer, and ignores those actions.       */
/* Inputs:      x,y,z - starting position on virtual terrain     */
/*              msktxt - 4 lines of text to be displayed at the  */
/*                       top of the screen                       */
/* Outputs:     mileage - distance travelled by user             */
/* Side effects:                                                 */
/*      This function must be declared in the calling routine    */
/*      as:                                                      */
/*              float navigate();                                */
/*****************************************************************/
#include "atrdefs.h"

extern int illegal[58][65];     /* bit map of illegal frames    */
extern char player;             /* currently active player      */
extern char nplayers;           /* number of active players     */
extern char mask[];             /* masked area over video       */
extern char window[];           /* portion of screen for text   */

float navigate(px,py,pz,msktxt)
char *msktxt[];
int *px,*py,*pz;
{
    unsigned frameno;           /* current video disc frame no. */
    unsigned calcframe();       /* declare calcframe function   */
    float readodom();           /* get elapsed mileage function */
    int x,y,z;                  /* coords of current position   */
    int oldx,oldy,oldz;         /* coords of previous position  */
    int sameframe;              /* flag to show no change       */
    int found;                  /* frame found on disc?         */
    int stat;                   /* input from joystick          */
    int try;                    /* number of attempts at frame  */


    /* start at parameters      */
    x = *px; y = *py; z = *pz;

    /* find frame on next player and switch video */
    player = toggle(player,nplayers);
    frameno = calcframe(x,y,z);
    for (try = 0; !findframe(player,frameno) && try < 2; try++);
    video(player);
    videoscr();

    /* display compass, odometer */
    clearw(mask,NORMAL);
    disptext(mask,msktxt,0,3);
    drawcompass(z);
    drawodom(x,y);
    center(window,window[3] - window[1]," Press button on joystick w
```

```
    while (TRUE)        /* let user drive till button hit */
        {

        /* save old coords in case new ones are illegal */
        oldx = x; oldy = y; oldz = z;

        /* reset flag indicating joystick movement      */
        sameframe = FALSE;

        /* calculate new coords from joystick input     */
        switch(stat = joystat())
            {
            case FIRE:  /* joystick button pressed      */
                videoln(window[3]);
                *px = x; *py = y; *pz = z;
                return(readodom());
                break;
            case FWD:
                travel(&x,&y,z,0);
                break;
            case REV:
                travel(&x,&y,z,8);
                break;
            case RIGHT:
                pivot(&z,15);
                break;
            case LEFT:
                pivot(&z,1);
                break;
            default:
                sameframe = TRUE;
                break;
            )

        /* if joystick in neutral position, restart loop        */
        if (sameframe)
            continue;

        /* if frame is illegal -- sound tone, restart loop      */
        if (getbit(illegal[x][y],z))
            {
            tone(ERRFREQ,ERRTIME);
            x = oldx; y = oldy; z = oldz;
            continue;
            )

        /* search for frame     */
        player = toggle(player,nplayers);
        frameno = calcframe(x,y,z);
        found = findframe(player,frameno);

        /* if error finding frame, restart loop */
        if (!found)
            (
```

```
            x = oldx; y = oldy; z = oldz;
            player = toggle(player,nplayers);
            continue;
            }

        /* if joystick position has changed, restart loop       */
        if (joystat() != stat)
             {
             x = oldx; y = oldy; z = oldz;
             player = toggle(player,nplayers);
             continue;
             }

        /* set up delays to smooth out travels  */
        switch(stat)
            {
            case FWD:
            case REV:
                if (x == oldx)  /* frames < 100 apart   */
                    nap(400);
                break;
            default:
                break;
            }

        /* switch the video to the current player       */
        video(player);

        /* update the status line on the screen         */
        compass(z);
        updtodom(x,y);

        } /* end of drive loop  */
}
```

```
/*****************************************************************/
/* Name:        navlssn                                          */
/* Description:                                                  */
/*      This function serves as the driver for a lesson in land */
/*      navigation training.  The lesson covers both terrain    */
/*      association and dead reckoning, and includes a practice */
/*      problem encompassing both techniques.                   */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

/* global variables     */
int illegal[58][65];            /* bit map of illegal frames    */
char player = 0;                /* currently active player      */
char nplayers = 1;              /* number of active players     */
char screen[] = {0,0,79,24};    /* screen definition            */
char window[] = {1,3,78,23};    /* portion of screen for text   */

main()
{
    static char *lssnmenu[] =   /* navigation training menu     */
        {
        "LAND NAVIGATION TRAINING",
        "Terrain Association",
        "Dead Reckoning",
        "Return to Main Menu"
        };
    int nchoices = 3;           /* number of menu choices       */
    int choice = 1;             /* menu option selected         */

    /* load illegal frame table */
    loadtab();

    /* display navigation introduction  */
    clearw(window,NORMAL);
    clearl(MSGLN,0,80,REVIDEO);
    disptxtf(window,"nltxt",0,2,18);
    drawcompass(0);
    resetodom(0,0);
    drawodom(0,0);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearw(window,NORMAL);
    delcompass();
    delodom();

    /* display menu of options - execute selected option        */
    while (choice = menu(window,lssnmenu,nchoices,choice))
        switch(choice)
            {
            case 1:     /* terrain association lesson & problem */
                trrnassn();
                break;
```

```
            case 2:     /* dead reckoning lesson & problem      */
                deadreck();
                break;
            case 3:     /* return to main menu                  */
                navsum();
                return;
                break;
            default:
                break;
            );
}
```

```
/*****************************************************************/
/* Name:        navrevu                                          */
/* Description:                                                  */
/*      This function serves as the driver for a review of       */
/*      map and compass skills needed for land navigation.       */
/*      If the student does not answer the questions in each     */
/*      skill area correctly, a list of references is provided   */
/*      to allow the student to relearn the skills.              */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      The text part of the screen is cleared on return         */
/*****************************************************************/
#include "atrdefs.h"

char *irtxt[] =                    /* introductory review text     */
(
"    ^[[34;47m                          S K I L L S   R E V I E W
    ^[[44;37m",
"",
"        This segment reviews the skills you will need for land navi
"",
"        You are given:",
"                          1) a map of the terrain",
"                          2) a black marker, and",
"                          3) a protractor.",
"",
"        Compass bearings appear on the bottom line of this screen."
);

char *refs[] =  /* list of references for prerequisite skills   */
(
"        Now you have reviewed the skills you need to navigate from
"    point to another.  For more information about topics covered, y
"    should consult these references:",
"",
"    ^[[34;47m    Topic                                        Reference
    ^[[44;37m",
"    Coordinate System                GTA 5-2-13: Pages 5-7,",
"                                      FM 21-26: Chap 3, Page 3-8, Par
"",
"    Azimuth Determination            GTA 5-2-13: Pages 17-21,",
"                                      FM 21-26: Chap 5, Page 5-1, Par
"",
"    Azimuth Conversion               GTA 5-2-13: Pages 21 -23,",
"                                      FM 21-26: Chap 5, Page 5-2, Par
"",
"    Distance Measurement             GTA 5-2-13: Pages 10-13,",
"                                     TEC Lesson 071-329-1008",
"                                      FM 21-26: Chap 4, Page 4-2, Par
"",
"    Self Location                    GTA 5-2-13: Pages 35-41,",
"                                      FM 21-26: Chap 5, Page 5-15, Pa
);

/* global variables     */
```

```
int illegal[58][65];            /* bit map of illegal frames    */
char player = 0;                /* currently active player      */
char nplayers = 1;              /* number of active players     */
char screen[] = {0,0,79,24};    /* screen definition            */
char window[] = {1,3,78,23};    /* portion of screen for text   */

main()
{
    int nogo = 0;               /* go/nogo for each review segment

    /* load illegal frame table */
    loadtab();

    /* text introduction for this review        */
    clearw(window,NORMAL);
    clearl(MSGLN,0,80,REVIDEO);
    disptext(window,irtxt,3,12);
    drawcompass(0);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    delcompass();
    clearw(window,NORMAL);

    /* go through each review segment   */
    nogo += coordsys();
    nogo += azimuth();
    nogo += distmeasure();
    nogo += resection();

    /* display references if at least one segment missed        */
    if (nogo > 0)
        {
        clearw(window,NORMAL);
        disptext(window,refs,0,19);
        center(window,window[3] - window[1],"Press button on joystic
e");
        joywait(FIRE);
        }
    clearw(window,NORMAL);
}
```

```
/*****************************************************************/
/* Name:        navsum                                           */
/* Description:                                                  */
/*      This function presents text summarizing the two methods */
/*      of land navigation.                                      */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

extern char window[];           /* useable portion of screen    */

navsum()
{
    /* land navigation summary */
    disptxtf(window,"nstxt",0,2,18);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearw(window,NORMAL);
    disptxtf(window,"nstxt",18,3,17);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearw(window,NORMAL);
}
```

```
/*****************************************************************/
/* Name:        navtest                                          */
/* Description:                                                  */
/*      This function serves as the driver for a series of test  */
/*      problems which the student does after taking the land    */
/*      navigation training.  Results of the land navigation     */
/*      test problems are measured according to how close to     */
/*      the destination the student comes.  Results are repor-   */
/*      ted as GO/NOGO for each problem.                         */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

char *ntsttxt[] =                       /* text intro to test proble
{
"    ^[[34;47m              L A N D   N A V I G A T I O N   T E S T
    ^[[44;37m",
"",
"        This test is designed to evaluate your ability to navigate
"    point to another in unfamiliar terrain.  You may use terrain as
"    dead reckoning, or a combination of the two methods.",
"",
"        There are three navigation problems in this test.  For each
"    we will place you at a location on the terrain which will be th
"    point.  We will give you its coordinates and the coordinates of
"    lease point.  You will plan a route from this start point to th
"    point using the map, protractor and marker, and then drive it.
"    free to take any route you wish, but your objective is to navig
"    best route to reach your destination within 50 meters.  When yo
"    you have arrived at the destination, press the button on the jo
"",
"        After you've completed all three of the problems, we'll dis
"    chart of results for each problem and an overall test result (G
"    to be entered in the Master Record.",
"",
"    ^[[34;47m        L A N D   N A V I G A T I O N   T E S T   R E
    ^[[44;37m",
"",
"          Student Name:                                  Attempt:",
"",
"              Problem     Distance from Release Pt.      Pass/Fail",
"              -------     -------------------------      ---------"
);

/* global variables     */
int illegal[58][65];            /* bit map of illegal frames    */
char player = 0;                /* currently active player      */
char nplayers = 1;              /* number of active players     */
char screen[] = (0,0,79,24);    /* screen definition            */
char window[] = (1,3,78,23);    /* portion of screen for text   */

main()
{
    char go_nogo;               /* result for test              */
```

```
    char name[80];              /* student name                  */
    int i;                      /* loop counter and index        */
    int try;                    /* test attempt number           */
    float howfar[4];            /* results of test problems      */

    /* load illegal frame table */
    loadtab();

    /* get password to continue */
    clearw(window,NORMAL);
    cursor(window[1] + 5,window[0] + 8);
    printf("Please have your intructor enter access password: ");
    if (!getpw("HAGGARD"))
        {
        clearw(window,NORMAL);
        return;
        }

    /* get student's name       */
    cursor(window[1] + 7,window[0] + 8);
    printf("Please enter student's name: ");
    getname(name);

    /* find out which attempt this is   */
    cursor(window[1] + 9,window[0] + 8);
    printf("Which attempt at the test is this (1/2/3)? ");
    try = gettry();

    /* display intro text */
    clearw(window,NORMAL);
    clearl(MSGLN,0,80,REVIDEO);
    disptext(window,ntsttxt,2,19);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* do appropriate problem set       */
    switch(try)
        {
        case 1:
            navtst1(howfar);
            break;
        case 2:
            navtst2(howfar);
            break;
        case 3:
            navtst3(howfar);
            break;
        default:
            break;
        };

    /* present results  */
    textscr();
    disptext(window,ntsttxt + 19,2,7);
    cursor(window[1] + 4,window[0] + 25);
    printf("%-25s",name);
```

```
    cursor(window[1] + 4,window[0] + 61);
    printf("%d",try);
    howfar[0] = 0;
    for (i = 1; i <= 3; i++)
        {
        cursor(window[1] + 7 + i,window[0] + 17);
        curs_off();
        printf("%d           %6.0f meters",i,howfar[i]);
        if (howfar[i] <= 50)
            {
            printf("%21s","PASS");
            howfar[0]++;
            }
        else printf("%21s","FAIL");
        }
    cursor(window[1] + 12,window[0] + 57);
    if (howfar[0] >= 2)          /* student passes       */
        {
        printf("^[[34;47m   GO   ^[[44;37m");
        go_nogo = 'G';
        }
    else
        {
        printf("^[[34;47m  NOGO  ^[[44;37m");
        go_nogo = 'N';
        }

    cursor(window[1] + 13, window[0] + 4);
    printf("Intructor,");
    cursor(window[1] + 14, window[0] + 8);
    printf("Please enter %c%d for Unit 3, Lesson 9 in this student's
rd.",go_nogo,try);
    cursor(window[1] + 16,window[0] + 8);
    if (go_nogo == 'N' && try < 3)
        printf("The test for this lesson may be taken %d more time(s
;

    /* get password to continue */
    do  {
        cursor(window[1] + 19,window[0] + 8);
        curs_off();
        printf("Please enter exit password: ");
        } while (!getpw("BLASCHE"));
    clearw(window,NORMAL);

}
```

```
/*****************************************************************/
/* Name:        navtst1                                          */
/* Description:                                                  */
/*      This function presents a set of three test problems to   */
/*      the student which test his land navigation ability.      */
/* Inputs:      None                                             */
/* Outputs:     dist - array of distances from release points    */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

#define X1       6      /* start point for first test problem   */
#define Y1      30
#define Z1      10

#define X2      28      /* start point for second test problem  */
#define Y2      39
#define Z2       0

#define X3      53      /* start point for third test problem   */
#define Y3      21
#define Z3      10

char *nt1txt[] =        /* text for first set of problems       */
{
"^[[34;47m                    N A V I G A T I O N   P R O B L E M   O
          ^[[44;37m",
"",
"      Your start point is 309824.  Use the map, compass and odomete
"    plan and navigate a route to your release point at 314802.",
"",
"^[[34;47m                    N A V I G A T I O N   P R O B L E M   T
          ^[[44;37m",
"",
"      Your start point is 282834.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 257862.",
"",
"^[[34;47m                    N A V I G A T I O N   P R O B L E M   T H
          ^[[44;37m",
"",
"      Your start point is 252813.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 264790.",
};

extern char window[];           /* useable portion of screen    */

navtst1(dist)
float dist[];
{
    int x,y,z;                  /* coordinates on terrain board */
    float navigate(), chkleg(); /* routines returning float     */

    /* present first problem    */
    x = X1; y = Y1; z = Z1;
    resetodom(x,y);
    navigate(&x,&y,&z,nt1txt);
```

```
    dist[1] = chkleg(&x,&y,&z,"314802","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
nd problem ");
    joywait(FIRE);
    delcoord();

    /* present second problem   */
    x = X2; y = Y2; z = Z2;
    resetodom(x,y);
    navigate(&x,&y,&z,ntltxt + 5);
    dist[2] = chkleg(&x,&y,&z,"257862","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
d problem ");
    joywait(FIRE);
    delcoord();

    /* present third problem    */
    x = X3; y = Y3; z = Z3;
    resetodom(x,y);
    navigate(&x,&y,&z,ntltxt + 10);
    dist[3] = chkleg(&x,&y,&z,"264790","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    delcoord();
)
```

```
/*****************************************************************/
/* Name:        navtst2                                          */
/* Description:                                                  */
/*      This function presents a set of three test problems to   */
/*      the student which test his land navigation ability.      */
/* Inputs:      None                                             */
/* Outputs:     dist - array of distances from release points    */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

#define X1       5      /* start point for first test problem   */
#define Y1      36
#define Z1       7

#define X2      43      /* start point for second test problem  */
#define Y2      26
#define Z2      10

#define X3      54      /* start point for third test problem   */
#define Y3      47
#define Z3       0

char *nt2txt[] =        /* text for first set of problems       */
(
"^[[34;47m                    N A V I G A T I O N   P R O B L E M   O
          ^[[44;37m",
"",
"      Your start point is 310831.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 306806.",
"",
"^[[34;47m                    N A V I G A T I O N   P R O B L E M   T
          ^[[44;37m",
"",
"      Your start point is 264819.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 284792.",
"",
"^[[34;47m                  N A V I G A T I O N   P R O B L E M   T H
          ^[[44;37m",
"",
"      Your start point is 251844.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 291862.",
);

extern char window[];           /* useable portion of screen    */

navtst2(dist)
float dist[];
(
    int x,y,z;                  /* coordinates on terrain board */
    float navigate(), chkleg(); /* routines returning float     */

    /* present first problem    */
    x = X1; y = Y1; z = Z1;
    resetodom(x,y);
    navigate(&x,&y,&z,nt2txt);
```

```
    dist[1] = chkleg(&x,&y,&z,"306806","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
nd problem ");
    joywait(FIRE);
    delcoord();

    /* present second problem   */
    x = X2; y = Y2; z = Z2;
    resetodom(x,y);
    navigate(&x,&y,&z,nt2txt + 5);
    dist[2] = chkleg(&x,&y,&z,"284792","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
d problem ");
    joywait(FIRE);
    delcoord();

    /* present third problem    */
    x = X3; y = Y3; z = Z3;
    resetodom(x,y);
    navigate(&x,&y,&z,nt2txt + 10);
    dist[3] = chkleg(&x,&y,&z,"291862","release poi t",50);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    delcoord();
)
```

```
/*****************************************************************/
/* Name:        navtst3                                          */
/* Description:                                                  */
/*      This function presents a set of three test problems to   */
/*      the student which test his land navigation ability.      */
/* Inputs:      None                                             */
/* Outputs:     dist - array of distances from release points    */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

#define X1      18      /* start point for first test problem   */
#define Y1      22
#define Z1       2

#define X2      55      /* start point for second test problem  */
#define Y2      31
#define Z2      13

#define X3      28      /* start point for third test problem   */
#define Y3      17
#define Z3      14

char *nt3txt[] =        /* text for first set of problems       */
{
"^[[34;47m                    N A V I G A T I O N   P R O B L E M   O
        ^[[44;37m",
"",
"      Your start point is 294814.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 269830.",
"",
"^[[34;47m                    N A V I G A T I O N   P R O B L E M   T
        ^[[44;37m",
"",
"      Your start point is 250824.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 276837.",
"",
"^[[34;47m                  N A V I G A T I O N   P R O B L E M   T H
        ^[[44;37m",
"",
"      Your start point is 282808.  Use the map, compass and odomete
"   plan and navigate a route to your release point at 314839.",
};

extern char window[];           /* useable portion of screen    */

navtst3(dist)
float dist[];
{
    int x,y,z;                  /* coordinates on terrain board */
    float navigate(), chkleg(); /* routines returning float     */

    /* present first problem    */
    x = X1; y = Y1; z = Z1;
    resetodom(x,y);
    navigate(&x,&y,&z,nt3txt);
```

```
    dist[1] = chkleg(&x,&y,&z,"269830","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
nd problem ");
    joywait(FIRE);
    delcoord();

    /* present second problem    */
    x = X2; y = Y2; z = Z2;
    resetodom(x,y);
    navigate(&x,&y,&z,nt3txt + 5);
    dist[2] = chkleg(&x,&y,&z,"276837","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
d problem ");
    joywait(FIRE);
    delcoord();

    /* present third problem     */
    x = X3; y = Y3; z = Z3;
    resetodom(x,y);
    navigate(&x,&y,&z,nt3txt + 10);
    dist[3] = chkleg(&x,&y,&z,"314839","release point",50);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    delcoord();
}
```

```
/*****************************************************************/
/* Name:        normal                                           */
/* Description:                                                  */
/*      This function sets the attributes for a row from col     */
/*      to col + n to normal video.                              */
/* Inputs:      row,col - starting coordinate for lowlighting    */
/*              n - number of characters to lowlight             */
/* Outputs:     None                                             */
/* Side effects:        cursor positioned off the screen        */
/*****************************************************************/
#include "atrdefs.h"

normal(row,col,n)
char row,col;
int n;
{
    register int i;
    register char a,c;
    register short ac;

    curs_off();
    for (i = 0; i < n; i++)
        {
        pcvscp(row,col + i);
        ac = pcvrca();
        c = ac & 0xff;
        a = NORMAL;
        pcvwca(1,c,a);
        }
}
```

```
/*****************************************************************/
/* Name:        drawodom,updtodom,resetodom,delodom,readodom     */
/* Description:                                                  */
/*      These functions provide a display of the user's          */
/*      position in the terrain.                                 */
/* Inputs:      x,y - coordinates in internal format             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      These routines all reference the mileage variable,       */
/*      which is global to these routines.                       */
/*****************************************************************/
#include "atrdefs.h"

static float mileage = 0;                /* distance traveled    */
static int x0,y0;                        /* previous coordinates */

updtodom(x,y)
int x,y;
{
    double sqrt();              /* square root calculator       */

    /* display updated distance traveled (meters) */
    mileage += 12 * sqrt((double) (x - x0) * (x - x0) + (y - y0) * (
    cursor(MSGLN,70);
    curs_off();
    printf("^[[47;34m%06.0f m^[[37;44m",mileage);

    /* save current coords      */
    x0 = x;
    y0 = y;
}

drawodom(x,y)
{

    /* display odometer label   */
    cursor(MSGLN,50);
    curs_off();
    printf("^[[47;34mDistance Traveled: ");

    /* display mileage  */
    updtodom(x,y);
}

resetodom(x,y)
int x,y;
{
    mileage = 0;
    x0 = x;
    y0 = y;
}

delodom()
{
    clearl(MSGLN,50,30,REVIDEO);
}
```

```
float readodom()
{
    return(mileage);
}
```

```
/*****************************************************************/
/* Name:        pivot                                            */
/* Description:                                                  */
/*      This routine is used to turn in one of 16 possible dir-  */
/*      ections on the ATR grid.                                 */
/* Inputs:      z - current bearing before turn is started       */
/* Inputs:      dir - direction in which to turn(RIGHT,LEFT)     */
/* Outputs:     z - new bearing after turn is completed          */
/* Side effects:                                                 */
/*****************************************************************/
int pivot(z, dir)
int *z,dir;
{
    *z = (*z + dir) % 16;
}
```

```
/*****************************************************************/
/* Name:        randloc                                          */
/* Description:                                                  */
/*      Generates a legal random location and direction on the   */
/*      terrain board.                                           */
/* Inputs:      px,py,pz - pointers to variables to be filled    */
/* Outputs:     px,py,pz - pointers to random location           */
/* Side effects:                                                 */
/*****************************************************************/

extern int illegal[58][65];     /* illegal frame table           */

randloc(px,py,pz)
int *px,*py,*pz;
{
    long time();        /* function returning elapsed seconds    */

    /* seed random number generator with time    */
    srand((unsigned) time());

    /* generate coordinates till legal  */
    do  {
        *px = (rand() % 54) + 2;
        *py = (rand() % 61) + 2;
        *pz = rand() % 16;
        ) while (getbit(illegal[*px][*py],*pz));
)
```

```
/*****************************************************************/
/* Name:        reject                                           */
/* Description:                                                  */
/*      This function blanks the video monitor and rejects all   */
/*      the video disc players.                                  */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"
#include "stdio.h"

extern char nplayers;           /* number of active players     */

reject()
{
    int player;

    /* blank the video monitor                    */
    video(BLANK);

    /* reject each player       */
    for (player = 0; player < nplayers; player++)
        {
        clearl(MSGLN,0,80,REVIDEO);
        printf("^[[34;47m   Rejecting videodisc player %d^[[44;37m",
        vdcmd(player,REJECT);
        )

    /* reset the multiplexor    */
    select(NULL);
}
```

```
/*****************************************************************/
/* Name:        resection                                        */
/* Description:                                                  */
/*      This function reviews the student's ability to locate    */
/*      himself in unfamiliar terrain.  It places him at a       */
/*      point in the terrain and allows him to pivot to shoot    */
/*      back azimuths.  He is allowed up to three attempts to    */
/*      correctly give his location.                             */
/* Inputs:      None                                             */
/* Outputs:     rc - 0 if located in <= three attempts           */
/*                 - 1 if not located                            */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

#define X       47              /* map location for resection   */
#define Y       32
#define Z        4

char *ltxt[] =                  /* text for resection review    */
(
"    ^[[34;47m                F I N D I N G   Y O U R   L O C A T I O
    ^[[44;37m",
"",
"        When you are navigating from one point to another, you need
"    how to locate your position by looking at the terrain around yo
"",
"        You will be placed somewhere on the terrain and asked to de
"    the coordinates of your location.  Use the joystick to move and
"    around.  You will be able to move up to 30 meters in any direct
"    study the surrounding terrain.  A tone will sound when you cann
"    any further in that direction.  Pressing the button on the joys
"    return you to your starting point."
);

extern char window[];           /* useable portion of screen    */

resection()
(
    int i;                      /* index and loop counter       */
    int x,y,z;                  /* coordinates on terrain board */
    int ntries;                 /* number of guesses made       */
    int rc;                     /* return code to calling pgm   */
    float dist;                 /* distance to actual location  */


    /* display location determination text      */
    clearw(window,NORMAL);
    disptext(window,ltxt,3,13);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* get user's response to coordinate question and evaluate */
    x = X; y = Y; z = Z;
```

```
    resetodom(x,y);
    if((ntries = askpos(&x,&y,&z,&dist)) <= 3)
        rc = 0;                    /* user guessed correctly       */
    else rc = 1;

    /* clear screen, message line and return    */
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    textscr();
    return(rc);
}
```

```
/*****************************************************************/
/* Name:        revideo                                          */
/* Description:                                                  */
/*      This function reverses video from col to col + n.        */
/* Inputs:      row,col - where to start                         */
/*              n       - how many colums to do                  */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

revideo(row,col,n)
char row,col;
int n;
{
    register int i;
    register char a,c;
    register short ac;

    curs_off();
    for (i = 0;i < n; i++)
        {
        pcvscp(row,col + i);
        ac = pcvrca();
        c = ac & 0xff;
        a = REVIDEO;
        pcvwca(1,c,a);
        }
}
```

```
/*****************************************************************/
/* Name:        drawlog,newlog                                   */
/* Description:                                                  */
/*      These functions are used to maintain a route log used    */
/*      with the dead reckoning method of land navigation.       */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      newlog() must be declared in calling routine as:         */
/*              struct logentry *newlog();                       */
/*****************************************************************/
#include "ctype.h"
#include "atrdefs.h"

char *logsheet[] =      /* blank log sheet                       */
(
"Öäää 1 ääääÔääää 2 äääÔäää 3 äääääÔäää 4 ääääÔäääää 5 äääääÔäää 6 ää
"·           · Measured ·  Actual   · Forward  ·              ·
"· Odometer · Distance · Distance · Magnetic · Declination ·  Grid
"· Reading  · (meters) · (meters) · Azimuth  · Correction  · Azimuth
"Çäääääääääéääääääääääéääääääääääéääääääääääéääääääääääéääääääää
"·           ·          ·          ·          ·              ·
"·           ·          ·          ·          ·              ·
"·           ·          ·          ·          ·              ·
"·           ·          ·          ·          ·              ·
"·           ·          ·          ·          ·              ·
"·           ·          ·          ·          ·              ·
"·           ·          ·          ·          ·              ·
"·           ·          ·          ·          ·              ·
"äääääääääêääääääääääêääääääääääêääääääääääêääääääääääêääääääää
);

char logwndw[] = (5,9,74,22);   /* window on screen for log      */

extern int offset;              /* declination correction        */

drawlog(p)                      /* display log sheet             */
struct logentry *p;
(
    int i;                      /* index and loop counter        */

    disptext(logwndw,logsheet,0,13);
    for (i = 0; i < 4; i++,p++)
        (
        if (isascii(p->point))
            (
            cursor(logwndw[1] + 5 + 2*i,logwndw[0] + 1);
            putch(p->point);
            );
        if (p->odom_rdg >= 0)
            (
            cursor(logwndw[1] + 5 + 2*i,logwndw[0] + 4);
            printf("%06.0f",p->odom_rdg);
            );
        if (p->m_dist >= 0)
            (
```

```
            cursor(logwndw[1] + 6 + 2*i,logwndw[0] + 14);
            printf("%4.0f",p->m_dist);
            };
        if (p->a_dist >= 0)
            {
            cursor(logwndw[1] + 6 + 2*i,logwndw[0] + 25);
            printf("%4.0f",p->a_dist);
            };
        if (p->mag_azmth >= 0)
            {
            cursor(logwndw[1] + 6 + 2*i,logwndw[0] + 37);
            printf("%3d°",p->mag_azmth % 360);
            cursor(logwndw[1] + 6 + 2*i,logwndw[0] + 50);
            printf((offset > 0) ? "+%d°" : " %d",offset);
            cursor(logwndw[1] + 6 + 2*i,logwndw[0] + 62);
            printf("%3d°",(p->mag_azmth + offset) % 360);
            };
        );
    curs_off();
}

struct logentry *newlog()       /* return pointer to empty log  */
{
    int i;                      /* index and loop counter       */
    struct logentry *p;         /* pointer to logentry          */
    static struct logentry log[4];      /* log of entries       */

    for (i = 0,p = &log[0]; i < 4; i++,p++)
        {
        p->point = -1;
        p->odom_rdg = -1;
        p->m_dist = -1;
        p->a_dist = -1;
        p->mag_azmth = -1;
        p->grd_azmth = -1;
        }
    return(p = &log[0]);
}
```

```
/*****************************************************************/
/* Name:        select                                           */
/* Description:                                                  */
/*      This function sends the prefix and configuration codes   */
/*      to select the given device to a Western Telematic CAS-   */
/*      41 code activated switch.  The prefix and configuration  */
/*      code definitions can be found in the "atrdefs.h" file.   */
/* Inputs:      device - device to select                        */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

select(device)
int device;
{
    /* clear serial buffer      */
    while (pcags(0) & 0x0100)
        pcarc(0);

    /* send prefix code and device code */
    pcawc(0,SELECT);
    pcawc(0,device);
}
```

```
/*****************************************************************/
/* Name:        sendfnum                                         */
/* Description:                                                  */
/*      This function converts an unsigned integer frame number */
/*      (i.e. < 65,535) to a character string and sends it, one */
/*      character at a time, to the videodisc player             */
/* Inputs:      player - which player to use                     */
/*              frameno - unsigned frame number                  */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

extern char vddigits[];          /* codes for numeric characters */

sendfnum(player,frameno)
char player;
unsigned frameno;
{
    char framestr[6];
    int i, len, sc;

    /* convert frame number to a string of digits       */
    len = stcu_d(&framestr,frameno,6);

    /* send digits one at a time to the player          */
    for (i = 0; i < len; i++)
        vdcmd(player,vddigits[framestr[i]-'0']);
}
```

```
/*****************************************************************/
/* Name:        shutdown                                         */
/* Description:                                                  */
/*      This function rejects the videodisc players and tells    */
/*      the user how to reset the switches for TICCIT.           */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

/* global variables     */
char nplayers = 1;                  /* number of active players     */
char screen[] = {0,0,79,24};        /* 25 x 80 screen               */
char window[] = {1,3,78,23};        /* useable portion of screen    */

main()
{
    int i;

    /* reject videodisc players */
    reject();

    /* show user TICCIT switch settings */
    clearw(window,NORMAL);
    clearl(MSGLN,0,80,REVIDEO);
    disptxtf(window,"swtxt2",0,3,14);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* clear keyboard buffer and screen */
    pckclr();
    clearw(screen,NORMAL);
    cursor(0,0);
}
```

```
/*****************************************************************/
/* Name:        startup(SONY LDP-1000A version)                  */
/* Description:                                                  */
/*      This function starts up the various components of the    */
/*      ATR system.  It puts up the main ATR screen, checks      */
/*      the frame table file and the link through the serial     */
/*      port to the switching device and spins up the players.   */
/*      It also displays the correct videodisc switch settings   */
/*      and allows the user to correct them if they are wrcng.   */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

/* global variables     */
char nplayers = 1;              /* number of active players     */
char screen[] = (0,0,79,24);    /* screen definition            */
char window[] = (1,3,78,23);    /* working area on screen       */

main()
(
    char player;                /* currently active player      */
    int fid;                    /* file identifier              */
    int i;                      /* index and loop counter       */
    int sc;                     /* status returned from port    */

    /* display main ATR screen  */
    textscr();

    /* frame table file on disk?        */
    if ((fid = open("badframe.atr",0)) == EOF)
        exit(2);                        /*      -emergency exit
    close(fid);

    /* init serial communications link  */
    initlink();

    /* blank the video monitor          */
    video(BLANK);

    /* show correct ATR switch settings */
    clearw(window,NORMAL);
    disptxtf(window,"swtxt",0,2,18);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearw(window,NORMAL);

    /* start players -- SONYs stop at first frame       */
    for (player = 0; player < nplayers; player++)
        (
        clearl(MSGLN,0,80,REVIDEO);
        printf("^[[34;47m   Starting videodisc player %d^[[44;37m",p
        while ((sc = vdcmd(player,PLAY)) == NULL)/* player not conne
            (
```

```
            cursor(MSGLN,0);
            curs_off();
            printf("^[[34;47m   Please reset left switch on player %
 and power it up again^[[44;37m",player + 1);
            }
        if (sc == ACK)                      /* wait for player to spin u
            while ((pcarc(0) >> 8) & 0x80)      /* ignore time-outs
                ;                               /*      - TICCIT onl
        else if (sc == NACK)                /* player spinning
            ;
        }
}
```

```
/*****************************************************************/
/* Name:        textscr                                          */
/* Description:                                                  */
/*      This function draws the screen for the ATR program.      */
/*      It is called at the beginning of the main program and    */
/*      each time text needs to be displayed on the screen.      */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

extern char screen[];           /* screen definition             */

textscr()
{

    /* display main ATR screen  */
    clearw(screen,NORMAL);
    video(BLANK);
    box2(screen);
    hline(screen[1] + 2,screen[0],screen[2] - screen[0]);
    center(screen,1,"A D V A N C E D  T E R R A I N  R E P R E S E
 N");
    clearl(MSGLN,screen[0],screen[2] - screen[0] + 1,REVIDEO);
}
```

```
/*****************************************************************/
/* Name:        time                                             */
/* Description:                                                  */
/*      This function reads the elapsed time from the system     */
/*      clock and return it in seconds.                          */
/* Inputs:      None                                             */
/* Outputs:     seconds - long integer                           */
/* Side Effects:                                                 */
/*      This function returns a long integer and must be         */
/*      declared as:                                             */
/*              long time();                                     */
/*****************************************************************/
#include "dos.h"

#define TICKS   1193180L
#define SECS      65536L

long time()
{
    union REGS regs;

    /* get number of elapsed ticks from system clock    */
    regs.h.ah = 0;              /* set interrupt entry point    */
    int86(0x1a,&regs,&regs);            /* DOS timer interrrupt */

    /* return number of seconds */
    return(((regs.x.cx << 16) + regs.x.dx) * SECS / TICKS);
}
```

```
/*****************************************************************/
/* Name:        toggle                                           */
/* Description:                                                  */
/*      This function switches the active player to the next     */
/*      player in the series.                                    */
/* Inputs:      curplayer - number of currently active player    */
/*              nplayers - total number of players in series     */
/* Outputs:     number of new active player                      */
/* Side effects:                                                 */
/*****************************************************************/
toggle(curplayer,nplayers)
int curplayer,nplayers;
{
    return((1 + curplayer) % nplayers);
}
```

```
/*************************************************************************
/* Name:        tone
/* Description:
/*      This function produces a tone from the speaker.
/* Inputs:      freq - frequency of tone (Hz)
/*              time - lenth of tone (sec/10)
/* Outputs:     None
/* Side effects:
/*************************************************************************
#define TIMERMODE       182     /* code to put timer in right mode
#define FREQSCALE   1190000L    /* basic time frequency in Hz
#define TIMESCALE     1230L     /* number of counts in 0.1 second
#define T_MODEPORT      67      /* port controls timer mode
#define FREQPORT        66      /* port controls tone frequency
#define BEEPPORT        97      /* port controls speaker
#define ON              79      /* signal to turn speaker on

tone(freq,time)         /* make tone of given frequency and length
int freq,time;
{
    int hibyte,lobyte,port;
    long i,count,divisor;

    /* set up frequency and time count  */
    divisor = FREQSCALE / freq; /* scale frequency to timer units
    lobyte = divisor % 256;
    hibyte = divisor / 256;
    count = TIMESCALE * time;   /* convert time to timer units

    /* set up timer(frequency) and speaker ports        */
    outp(T_MODEPORT,TIMERMODE); /* prepare timer for input
    outp(FREQPORT,lobyte);      /* set low byte of time register
    outp(FREQPORT,hibyte);      /* set high byte of time register
    port = inp(BEEPPORT);       /* save port setting

    /* sound speaker for desired length of time */
    outp(BEEPPORT,ON);          /* turn on speaker
    for (i = 0; i < count; i++)
        ;                       /* mark time
    outp(BEEPPORT,port);        /* turn off speaker, restore setting
}
```

```
/*****************************************************************/
/* Name:        travel                                           */
/* Description:                                                  */
/*      This function is used to travel from grid center to      */
/*      grid center in the ATR grid.  All travel is forward or   */
/*      backward; turns are done as a pivot and then a travel.   */
/* Inputs:      x,y,z - current location and direction           */
/*              dir - direction of travel (FWD,REV)              */
/* Outputs:     x,y - new location on grid after traveling       */
/* Side effects:                                                 */
/*****************************************************************/
travel(x, y, z, dir)
int *x,*y,z,dir;
(
    switch((z + dir) % 16)
        (
        case 0: (*y)++;
                break;
        case 1: (*x)++;
                (*y) += 2;
                break;
        case 2: (*x)++;
                (*y)++;
                break;
        case 3: (*x) += 2;
                (*y)++;
                break;
        case 4: (*x)++;
                break;
        case 5: (*x) += 2;
                (*y)--;
                break;
        case 6: (*x)++;
                (*y)--;
                break;
        case 7: (*x)++;
                (*y) -= 2;
                break;
        case 8: (*y)--;
                break;
        case 9: (*x)--;
                (*y) -= 2;
                break;
        case 10: (*x)--;
                 (*y)--;
                 break;
        case 11: (*x) -= 2;
                 (*y)--;
                 break;
        case 12: (*x)--;
                 break;
        case 13: (*x) -= 2;
                 (*y)++;
                 break;
        case 14: (*x)--;
                 (*y)++;
```

```
                break;
        case 15: (*x)--;
                (*y) += 2;
                break;
        default: break;
        }
}
```

```
/*****************************************************************/
/* Name:        trrnassn                                         */
/* Description:                                                  */
/*      This function introduces terrain association as a tech-  */
/*      nique for land navigation.  The list of steps shown      */
/*      is covered one by one as part of an example.  A sample   */
/*      problem is also provided to allow the student to prac-   */
/*      tice immediately following the instruction.              */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

extern char window[];           /* portion of screen for text   */

trrnassn()
{
    /* terrain association intro        */
    clearw(window,NORMAL);
    clearl(MSGLN,0,80,REVIDEO);
    disptxtf(window,"tatxt",0,3,13);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* planning the route       */
    clearw(window,NORMAL);
    disptxtf(window,"rptxt",0,3,18);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* go throught the route plan       */
    detazmth();
    checkpts();

    /* do example problem       */
    trrnxamp();

    /* let user do practice problem     */
    trrnprob();

    /* closing text on terrain association      */
    disptxtf(window,"tctxt",0,3,15);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);
    clearw(window,NORMAL);

)
```

```
/*****************************************************************/
/* Name:        trrnprob                                         */
/* Description:                                                  */
/*      This function provides a student with a problem to       */
/*      practice land navigation by terrain association. He is   */
/*      given a start point, a release point and several inter-  */
/*      mediate checkpoints. He must navigate from the start     */
/*      point to the release point, passing within 50 meters of  */
/*      each checkpoint, which he indicates by pressing the      */
/*      button on the joystick. If he is not on track, he is     */
/*      placed at the checkpoint where he should be and can      */
/*      continue.                                                */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "stdio.h"
#include "atrdefs.h"

#define X       21              /* coords of starting location  */
#define Y        2
#define Z        0

int tplst[] =           /* driving instructions for problem     */
    {HALT, HALT};

char *tptxt2[] =                /* text for practice problem    */
{
"       You have planned your route and you're ready to go. You mus
"    to each checkpoint in the correct order.  You should be able to
"    one within 30 meters.  A compass and odometer on the bottom lin
"    screen will show magnetic azimuths and distance traveled.",
"",
"       You are now at your start point at 291790.  As the first le
"    route you need to navigate to the first checkpoint, the depress
"    298812.  You may start whenever you are ready.",
"",
"",
"       Your next checkpoint is the depression at 298823.  You are
"    travel whenever you are ready.",
"",
"",
"       The next checkpoint is the road at 304832.  You may start w
"    you are ready.",
"",
"",
"       Your release point is the boulder at 308841. You may start
"    you are ready.",
"",
"      Great!  You made it to the release point without missing a ch
"    You have learned to navigate well using terrain association.  If
"    comfortable navigating by terrain association, you are ready to
"    on to the section on navigation by dead reckoning.",
"",
"      Good!  You made it to the release point successfully.  Althou
"    route went a little astray from the checkpoints, you have learne
```

```
"   igate fairly well using terrain association.  If you are comfort
"   your ability in terrain association, you should go on to dead re
"",
"      You seem to have had some difficulty navigating to the releas
"   You should review this section on terrain association again befo
"   on to the dead reckoning section.  You may also want to practice
"   on the terrain using the Free Travel option from the Main Option
};

extern char window[];              /* useable portion of screen    */
extern char mask[];                /* mask over top part of video  */

trrnprob()
(
    int i;                         /* index and loop counter       */
    int x,y,z;                     /* coordinates on terrain       */
    float dist[5];                 /* distance from checkpoints    */
    float navigate(),chkleg();     /* routines returning float     */

    /* explain practice problem */
    clearw(window,NORMAL);
    disptxtf(window,"tptxt",0,3,9);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* display route plan       */
    textscr();
    disptxtf(window,"plntxt",0,3,17);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* show checkpoint coordinates      */
    clearw(window,NORMAL);
    disptxtf(window,"ckptxt",0,3,16);
    center(window,window[3] - window[1],"Press button on joystick to
    joywait(FIRE);

    /* let user do practice problem     */
    x = X; y = Y; z = Z;
    resetodom(x,y);
    autodriv(&x,&y,&z,tplst);
    clearw(mask,NORMAL);
    disptext(mask,tptxt2,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* first leg        */
    navigate(&x,&y,&z,tptxt2 + 5);
    dist[1] = chkleg(&x,&y,&z,"298812","depression",30);
    center(window,window[3] - window[1]," Press button on joystick t
 leg ");
    joywait(FIRE);
    delcoord();

    /* second leg       */
```

```
    navigate(&x,&y,&z,tptxt2 + 9);
    dist[2] = chkleg(&x,&y,&z,"298823","depression",30);
    center(window,window[3] - window[1]," Press button on joystick t
 leg ");
    joywait(FIRE);
    delcoord();

    /* third leg        */
    navigate(&x,&y,&z,tptxt2 + 13);
    dist[3] = chkleg(&x,&y,&z,"304832","road",30);
    center(window,window[3] - window[1]," Press button on joystick t
 leg ");
    joywait(FIRE);
    delcoord();

    /* last leg         */
    navigate(&x,&y,&z,tptxt2 + 17);
    dist[4] = chkleg(&x,&y,&z,"308840","boulder",50);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    videoln(window[3]);
    clearw(mask,NORMAL);

    /* evaluate performance and present appropriate screen      */
    dist[0] = 0;
    for (i = 1; i <= 4; i++)
        if (dist[i] <= 50)
            dist[0]++;
    if (dist[0] == 4)                    /* made all checkpoints */
        disptext(mask,tptxt2 + 21,0,3);
    else if (dist[4] <= 50 && dist[0] >= 2)
        disptext(mask,tptxt2 + 26,0,3);
    else
        disptext(mask,tptxt2 + 31,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    textscr();
}
```

```
/*****************************************************************/
/* Name:        trrnxamp                                         */
/* Description:                                                  */
/*      This function takes the student through an example of    */
/*      land navigation using the techniques of terrain asso-    */
/*      ciation, by driving through a sample problem under       */
/*      program control while giving textual comments.           */
/* Inputs:      None                                             */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

#define X       54              /* coords of starting location  */
#define Y       60
#define Z        8

int txlist0[] =         /* lists of driving instructions        */
{HALT, HALT};

int txlist1[] =
{FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, HALT};

int txlist2[] =
{RIGHT, FWD, LEFT, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD,
 HALT};

int txlist3[] =
{LEFT, LEFT, LEFT, LEFT, LEFT, FWD, FWD, FWD, FWD, RIGHT, HALT};

int txlist4[] =
{FWD, LEFT, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, FWD, H

char *txtxt[] =                         /* text for navigation examp
{
"       Now we'll travel from point A to point B along the route we
"    stopping at each checkpoint.  A compass and odometer on the bot
"    of this screen will show magnetic bearings and distance travele
"",
"      We are at point A, facing south.  Notice the contour lines on
"   that cross the route you have drawn.  They show that we will be
"   downhill.  Let's go to the first checkpoint, the tree at 251848.
"",
"      We're at the first tree at 251848.  This tree is a good check
"   cause it is at the bottom of a hill and at the north end of a va
"   leys are linear features, the best type of checkpoint.  Now, com
"   distance displayed on the odometer with your measureed distance.
"",
"      Let's continue to our next checkpoint, the tree at the south
"   valley (250831 on the map).  We choose to travel along the valle
"   movement is easiest along valleys and ridge crests.  We will tak
"   turn at first, but our general direction of travel will be to th
"",
"      Now we're at the next tree.  See how your distance measuremen
"   with the odometer reading below.  Your next checkpoint is to the
"   paved road at 255831.  A road is another good linear feature to
```

```
"   checkpoint.  Note that the contour lines show that we will be go
,
"",
"      Here's the road.  So far, we've been using linear features, w
"   the best checkpoints. Elevation changes, such as hills or depres
,
"   make good checkpoints.  Our release point is the depression towa
,
"   at 270831.  Check the odometer reading, and when you're ready, w
"",
"      We did it!  We're at the depression at 270831 on your map.  L
"   the odometer below, we have traveled a total of 486 meters.  Thi
"   should be close to the distance you measured earlier.",
};

extern char mask[];             /* masked area of video screen  */
extern char window[];           /* useable portion of screen    */

trrnxamp()
{
    int x,y,z;                  /* coordinates on terrain       */


    /* show terrain at start point */
    x = X; y = Y; z = Z;
    resetodom(x,y);
    autodriv(&x,&y,&z,txlist0);
    clearw(mask,NORMAL);
    disptext(mask,txtxt,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    videoln(window[3]);

    /* get ready to drive to first checkpoint   */
    clearw(mask,NORMAL);
    disptext(mask,txtxt + 4,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,txlist1);

    /* at first checkpoint      */
    clearw(mask,NORMAL);
    disptext(mask,txtxt + 8,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    videoln(window[3]);

    /* get ready to drive to next checkpoint    */
    clearw(mask,NORMAL);
    disptext(mask,txtxt + 13,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
```

```
    autodrive(&x,&y,&z,txlist2);

    /* at second checkpoint; prepare to go to third one */
    clearw(mask,NORMAL);
    disptext(mask,txtxt + 18,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,txlist3);

    /* at third checkpoint; prepare to go to release point      */
    clearw(mask,NORMAL);
    disptext(mask,txtxt + 23,0,3);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);
    autodrive(&x,&y,&z,txlist4);

    /* at release point; check odometer for total distance traveled.
    clearw(mask,NORMAL);
    disptext(mask,txtxt + 28,0,2);
    center(window,window[3] - window[1]," Press button on joystick t
);
    joywait(FIRE);

    /* clear window, message line */
    textscr();
}
```

```
/*****************************************************************/
/* Name:        vdcmd                                            */
/* Description:                                                  */
/*      This function sends a single command to the indicated    */
/*      SONY LDP-1000A videodisc player and returns the ACK or   */
/*      NACK from player to the calling routine.                 */
/* Inputs:      player - which player to send command            */
/*              command - 8-bit command to the player            */
/* Outputs:     sc - return code from player                     */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

vdcmd(player,command)
char player;
char command;
{
    int sc;
    int r,c;

    /* select player    */
    select(VDBASE + player);

    /* clear buffer     */
    while (pcags(0) & 0x0100)
        pcarc(0);

    /* send command     */
    pcawc(0,command);

    /* wait for player to ACK/NAK command       */
    sc = pcarc(0);

    /* return ACK/NAK from player       */
    return(sc & 0xff);
)
```

```
/*****************************************************************/
/* Name:        video                                            */
/* Description:                                                  */
/*      This function displays video from the given player,      */
/*      using a DDI serial video switch, and adjusts the TICCIT  */
/*      color board control registers for sync and interlace.    */
/* Inputs:      player - which player's video to display         */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*****************************************************************/
#include "atrdefs.h"

video(player)
char player;
{

    /* select video switch      */
    select(VIDEO);

    /* switch video selector to this player     */
    if (player == BLANK)
        {
        pcawc(0,player);
/*      outp(CRTCINDEX,8);              */
/*      outp(CRTCDATA,INTERLACE);       */
/*      outp(VIDEOCTRL,INTSYNC);        */
        }
    else
        {
        pcawc(0,VIDBASE | (player << 0x01));
/*      outp(CRTCINDEX,8);              */
/*      outp(CRTCDATA,NONINTERLACE);    */
/*      outp(VIDEOCTRL,EXTSYNC);        */
        }
}
```

```
/*****************************************************************/
/* Name:        videoscr, videoln                                */
/* Description:                                                  */
/*      These routines display a video background using the      */
/*      TICCIT video board.  This is done by making the fore-    */
/*      ground black (transparent) for each group of 10 pixels   */
/*      and setting the corresponding attribute byte to display  */
/*      the video in the background.                             */
/* Inputs:      None for videoscr                                */
/*              line for videoln                                 */
/* Outputs:     None                                             */
/* Side effects:                                                 */
/*      These functions are not flexible...they clear only       */
/*      the 24 x 80 screen or a full 80 char line                */
/*****************************************************************/
#include "atrdefs.h"

char buf[] =                        /* 10 pixel + attribute byte    */
    {0,0,0,0,0,0x80);

videoscr()
{
    register int seg,off;           /* video memory segment:offset  */
    unsigned n;                     /* number of bytes to change    */


    /* display video bg for 24 x 80 screen      */
    n = sizeof(buf);
    for (seg = 0xd280; seg < 0xea80; seg += 0x40)
        for (off = 0x20; off < 0x320; off +=0x8)
            poke(seg,off,buf,n);

}

videoln(line)
char line;
{
    int lntop,lnbot;                /* top and bottom row of pixels */
    int seg,off;                    /* video memory segment:offset  */
    unsigned n;                     /* number of bytes to change    */

    /* calculate line addresses         */
    lntop = 0xd280 + line * 0x100;
    lnbot = 0xd280 + (line + 1) * 0x100;

    /* display video bg for 80 char line        */
    n = sizeof(buf);
    for (seg = lntop; seg < lnbot; seg += 0x40)
        for (off = 0x20; off < 0x320; off +=0x8)
            poke(seg,off,buf,n);

}
```

APPENDIX C

SONY LDP-1000A MANUAL

SONY

VIDEODISC PLAYER

LDP-1000A

OPERATING INSTRUCTIONS

Before operating the unit, please read this manual thoroughly and retain it for future reference.

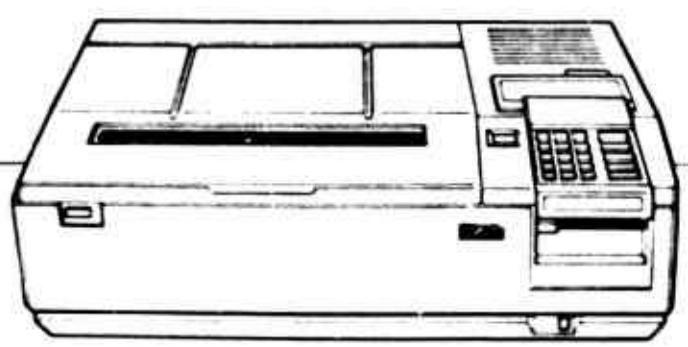


TABLE OF CONTENTS

OWNER'S RECORD

The model and serial numbers are located at the rear. Record the serial number in the space provided below. Refer to these numbers whenever you call upon your Sony dealer regarding this product.

Model No. LDP-1000A Serial No. 50356

WARNING

To prevent fire or shock hazard, do not expose the set to rain or moisture.

To avoid electrical shock, do not open the cabinet. Refer servicing to qualified personnel only.

CAUTION

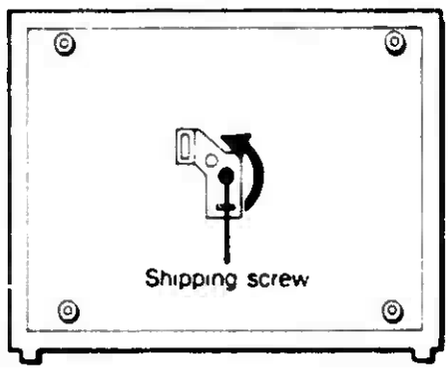
Use of controls or adjustments or performance of procedures other than those specified herein may result in hazardous radiation exposure.

PRECAUTIONS

- This mechine is designed for operation in e horizontal posifion end only with e 120 V ec, 60 Hz power supply.
- Do not Instalt the set in e locetion near heat sources, such es redietors or eir ducts, or In a piece subject to direct sunlight, excessive dust, mechanicel vibretion or shock.
- Allow edequete eir circuletion to prevent internal heat buildup. Do not place the set on surfeces (rugs, blankets, etc.) or neer materials (curtains, draperies) that may block the ventiletion holes.
- After playing a disc, remove it from the compartment If the set will not be used for any length of time. Do not trensport the set with e disc In piece.
- To disconnect the cord, pull it out by the plug. Never putl the cord Itself.
- Should eny solid object or liquid fatl into the cabinet, unplug the set and have it checked by qualified personnel betore operating it eny further.
- Do not operate the set right after having transported it from e cold tocation directly to a warm location or in e room whose temperature rises suddenly, because moisture may condense in the operating section of the set. Wait for about en hour before turning the power on in the new location or tet the room temperature rise gradually.
- Clean the cabinet, panel and controls with a dry soft cloth, or soft cloth lightly moistened with a mild detergent solution. Do not use any type of solvent such as alcohol or benzine which may damage the finish.
- Remove the cushion end be sure to save it. When the LDP-1000A is transported or when shock to the compartment lid is expected, be sure to attach the cushion.
- Save the original shipping carton and packing material; they will come in handy it you ever have to ship your set. For maximum protection, repack the set as it was originally packed at the tactory.
- If you have any questions about this machine, contact your dealer or your nearest Sony authorized service facility.

LOOSEN THE SHIPPING SCREW UNDERNEATH THE PLAYER.

The shipping screw is screwed down et the factory to secure the mechanism inside the ptayer.
Be sure to turn the shipping screw counterclockwise with a coin or similar object until the screw is loose soon efter unpacking.



A spiral pattern of pits is recorded about 1.1 mm under the surface of the video disc.
In the LDP-1000A videodisc player, a laser beam focuses on the pits and then reflects. Variations in the reflected beam are detected and converted into the video and audio playback signals. The playback picture and sound are obtained by a monitor or a TV receiver connected to the video disc player.
This videodisc system has the following features:

No physical contact between pick-up system and disc
Because a laser beam is employed for signal pick-up, there is no physical contact with the disc, which means no wear. In addition, because the pit pattern is recorded below the surface of the disc, it is not necessary to be constantly on guard against fingerprints and dust, making the video disc easy to handle.

High accuracy insured high-quality picture
Newly-developed CCD chips enable the effects of the time base error and disc eccentricity to be eliminated.

Flexible use of video discs
The brushless spindle motor makes the system set in the standby mode in about 10 seconds after a disc is inserted. The system can play both the CAV* and CLV** video discs. With a CAV disc, the system can play back (or in reverse) at various speeds, can display a still picture and can perform a number of other functions with the aid of a built-in microprocessor.

High speed access
You can locate a particular point on a disc within five seconds. With a CAV disc, a particular frame or segment can be searched for, and with a CAV/CLV disc with chapter data, a particular chapter can be searched for.

• CAV (constant angular velocity) disc
The CAV disc rotates at 1800 r.p.m. and the laser beam moves from the inner part of the disc to the outer. Up to 30 minute playback is possible on one side of the disc. On the disc, up to 54,000 frames can be recorded. Each frame of the playback picture is recorded as one rotation and the frame number is recorded on the track.
With a CAV disc, playback at variable speeds, frame number and play mode display and operations with control instructions are possible.

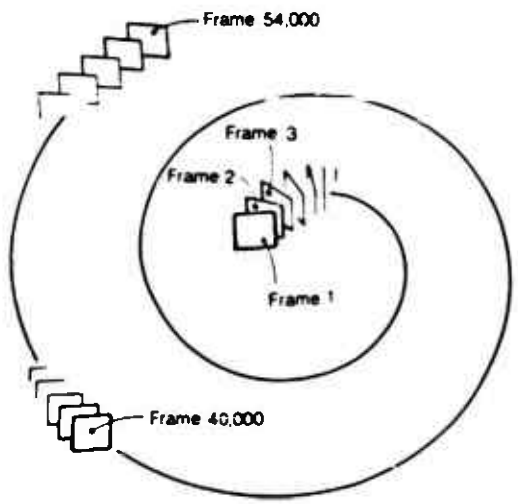
• • CLV (constant linear velocity) disc
The CLV disc rotates at a speed between 1800 r.p.m. and 700 r.p.m. with constant linear velocity. The laser beam moves from inner part of the disc to the outer as with CAV disc. Playback of up to one hour is possible on one side of the disc, though only the normal play, the scan and the search operations are possible. The elapsed playback time can be displayed on the monitor screen.

Remote control of the system is possible
The system's control unit can be detached and used as a wireless remote control unit or with the supplied remote cable, as a wired remote control unit.

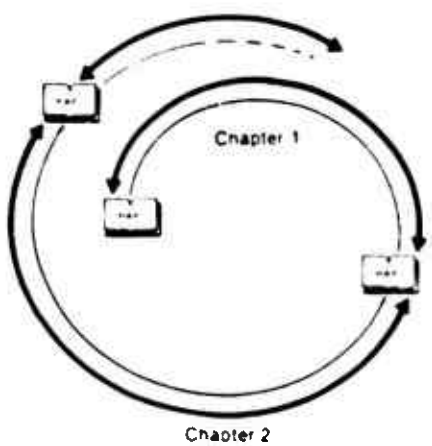
A variety of possible connections
The LDP-1000A is equipped with an RS232C serial interface connector so that it can be connected to a computer. The LDP-1000A is also equipped with SYNC IN and SC IN connectors so that it can be operated in synchronization with the external sync signal and so that a special effects generator can be connected to the LDP-1000A.

DATA ON THE VIDEO DISCS

Frames
A CAV disc has up to 54,000 "frames" which are numbered in sequence. You can search for a particular frame or repeat a particular sequence of frames.



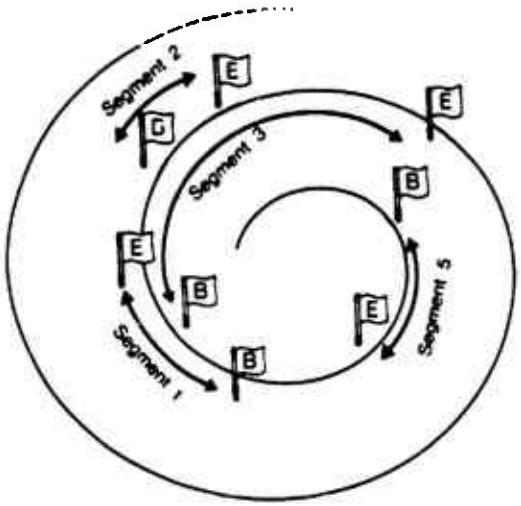
Chapters
There are CAV and CLV discs on which "chapters" are recorded, as the chapter of a book. If a chapter number is displayed after a frame number has been displayed (on a CAV disc) or if playback time is displayed in minutes (on a CLV disc) when you press the [INDEX] key, the disc has chapter data.
You can easily search for a particular chapter and play it back repeatedly.



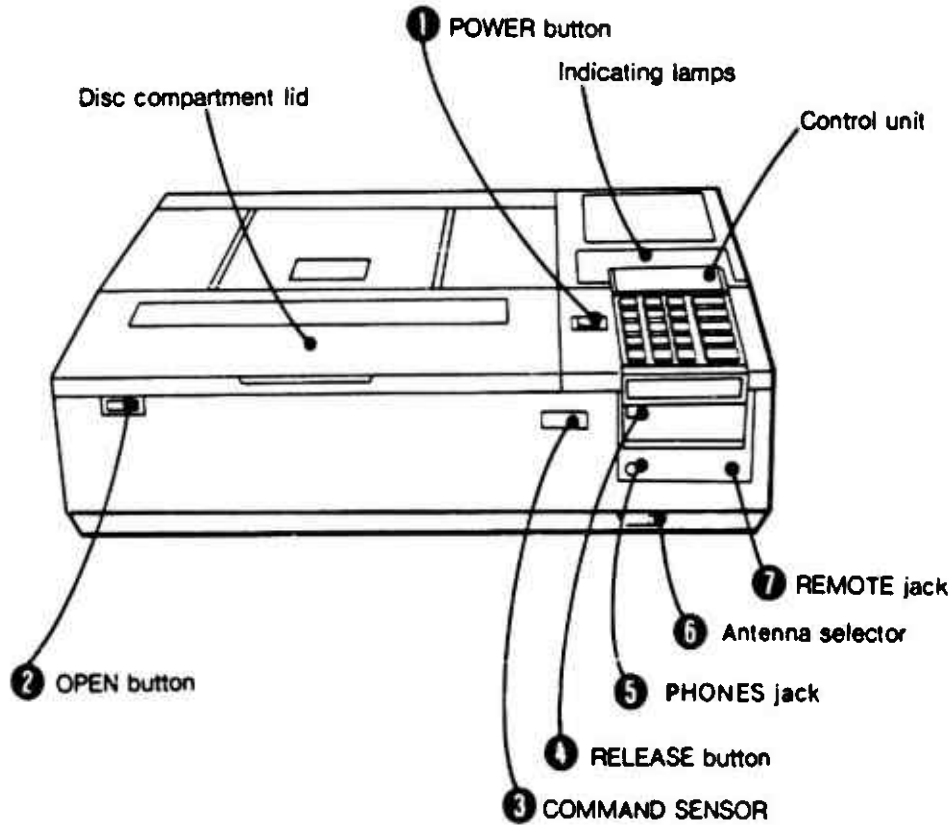
Segments

Using the frame numbers on e CAV disc, you can divide the disc into up to 63 seperete segments by designating the beginning and end freme of a aequence.

You can search for a perticuiar segment end play it back repeatedly.



FRONT PANEL



1 POWER button
Press to turn on the power of the player.
To turn off the power, press the button again.

2 OPEN button
Press to open the disc compartment lid for insertion or removal of the video disc.

3 COMMAND SENSOR
The red COMMAND SENSOR lamp will blink to show that the player detects that a key of the control unit has been pressed.

4 RELEASE button
Press this button while detaching the control unit.

5 PHONES jack
Connect headphones here to monitor the audio.

6 Antenna selector
This selector selects the picture on the monitor screen. When you want to watch the program from the disc, set this selector to VDP. When you want to watch the TV program from the antenna, set this selector to ANT.

7 REMOTE jack (special mini jack)
Connect to the remote jack on the control unit with the remote cable when the control unit is to be used as a wired remote commander.

CONTROL UNIT

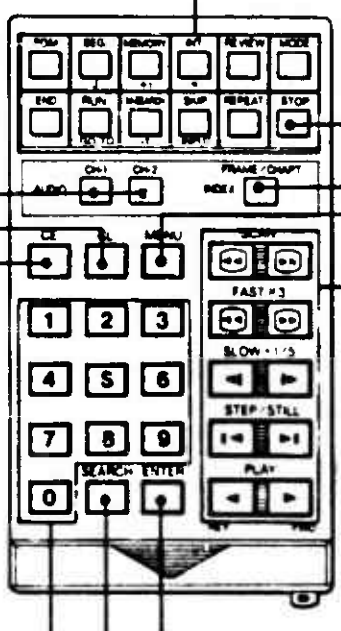
- The ☐ keys cannot function when using a CLV disc.

The [CE] kay cleers tha iest dats entered.
Press this key when you press an incorrect kay.

The [CL] kay is the all-claar key. Press this key when you wsnt to cancal an operation using the control instruction kays, such es entering the control instructions, operation with control Instructions or check ot the entered control instructions.

AUDID CH-1 and CH-2 keya
Esch lsser disc hes two eudio channels, chsnnel 1 snd 2. When tha player is turned on, both audio chennals are turned on. The AUDIO CH-1 and CH-2 lamps witl light. To turn the audio channel 1 or 2 off, press the CH-1 or CH-2 kay. The appropriate indicating lamp will go off. Pressing the key again turns the sudio channel on.

Control inatruction kaya
Thase keys sre used for tha operstion with control instructicns —memorizing the segmant dsts, memorizing the controt instructions, operating the disc with control instructions and so on.



STOP kay
Press this key to stop displsying the picture. When anothar key is pressed, the desired playback mode will start at the point where the STOP kay was pressed.

INDEX kay
Press to display the play mode, frame number, chapter number and data tor control instructions on the monitor screen while playing the CAV disc. When the CLV disc is playing, the time elapsed since the beginning ot the program will be displayed, and then chapter number will be displayed if it is recorded. Press this key again to turn the display oft.

MENU kay
Press this key to see the menu again during playing a CAV disc.

Function kaya
SCAN key
Keep holding this key down tor high-speed (about 100 times ot normsl spead) playback. Whan you releasa this key, playback will be resumed at normal speed.
FAST key
Press this key tor tripple-speed playback.
SLOW kay
Press this key for slow-speed (1/5 ot normal speed) playback.
STEP/STILL kay
Press this key for a still picture. tt you press this kay again and immedistely release it, the next trame will be displsyed. It you keep the kay dapressad, the picture will advance trama-by-trama.
PLAY kay
Press this kay tor normsl speed playback.

Figure keys

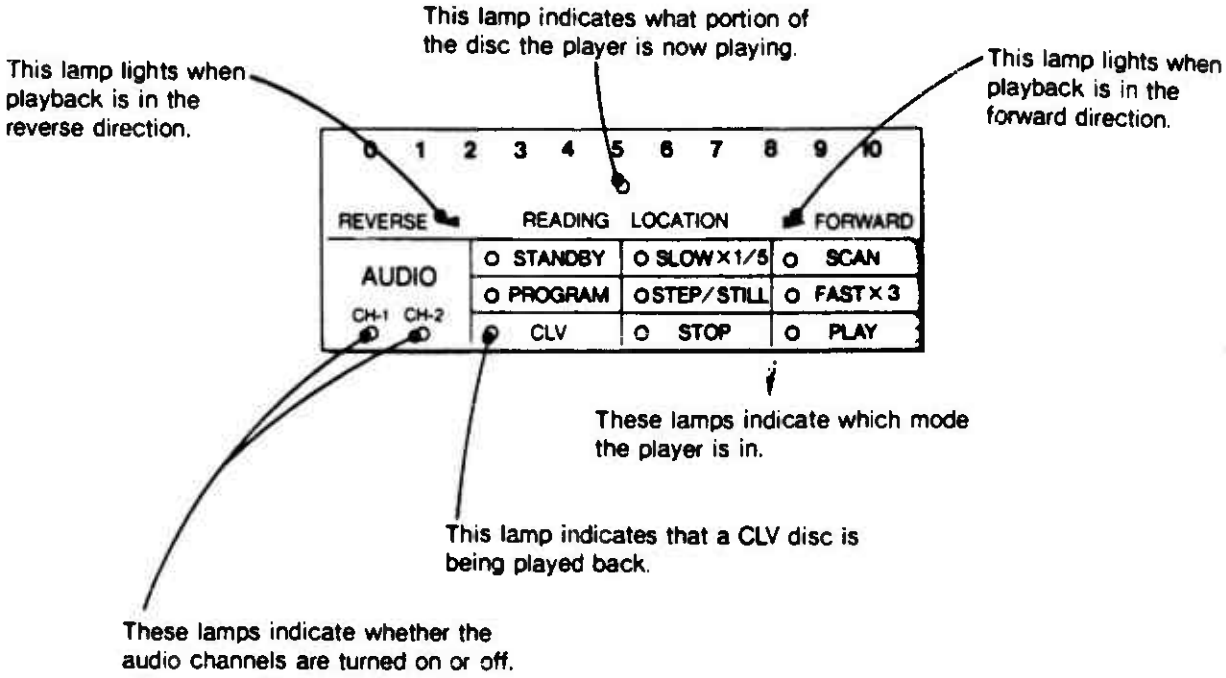
SEARCH key
Press this key tor the search operation

ENTER kay
Press this key to advanca the step to the next during memorizing the segment data or the control instructions or to end the above mentioned operation.

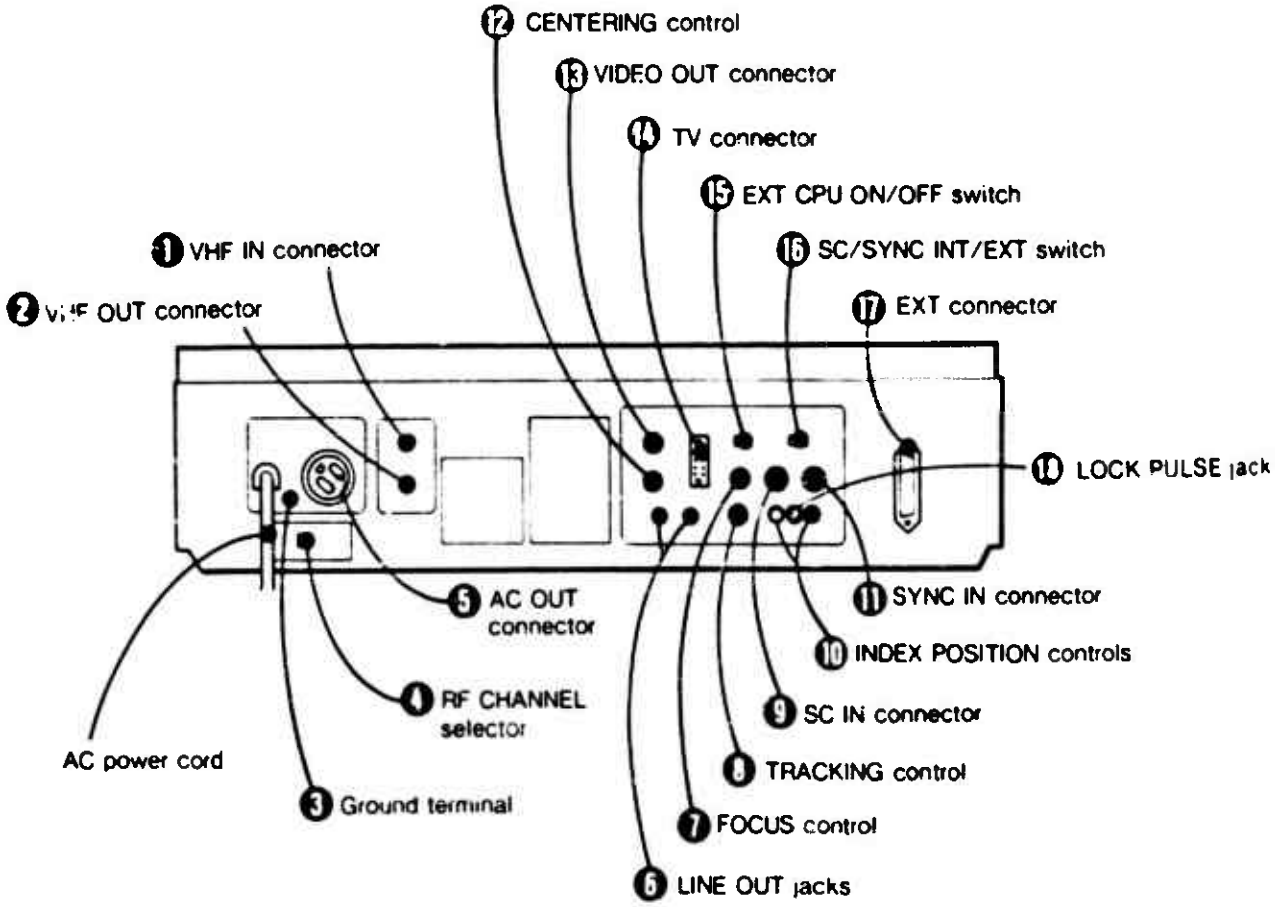
Notaa on tha function keya

- The right key of the two keys [▶] is for playback and the left key [◀] is for reverse.
- The audio can be monitored only in the forward play mode.
- Press the desired key when you want, no matter what mode the player is in. The player enters into the desired mode.

INDICATING LAMPS



REAR PANEL



❶ VHF IN connector (F type connector)
Connect the 75-ohm coaxial cable for VHF antenna here. If the antenna cable is a 300-ohm twin lead, use the EAC-25 external antenna connector (optional).

❷ VHF OUT connector (F type connector)
Connect to the VHF antenna connector of the TV receiver. Either of the following signals, selected by the antenna selector on the front, can be supplied.
— VHF TV signal connected to the VHF IN connector
— the program on the disc (which is converted into the VHF signal by the built-in RF unit)

❸ Ground terminal [⏚]
To reduce hum, connect this terminal to an earth ground with a ground wire.

❹ RF CHANNEL* selector
Selects the channel to which the output signal of the VHF OUT connector is fed. Set the selector to channel 3 or 4, which is not active in your area.
* RF (Radio Frequency) channel: The built-in RF unit modulates the playback signal of the disc into the frequency of the VHF channel 3 or 4, which we call the RF channel, so that the picture from the disc can be displayed by the TV receiver.

❺ AC OUT (outlet) connector
This outlet supplies ac power to other video equipment whose power consumption is no more than 400 watts. Power is supplied to the connected equipment regardless of the position of the player's POWER switch.

❻ LINE OUT jacks (phono jacks)
Connect to the line input jacks of audio equipments.

❼ FOCUS control
Normally set this control at the center detent position.
If a picture with noise appears on the monitor screen, turn this control until you get the best possible picture.
After playing this particular disc, return this control to its center detent position.

❽ TRACKING control
Normally set this control at the center detent position.
If a picture with noise appears on the monitor screen, first turn the FOCUS control and then this control until you get the best possible picture.
After playing this particular disc, return this control to its center detent position.

❾ SC IN connector (BNC connector)
Connect an external sync generator. The connector accepts 3.58 MHz subcarrier.

❿ INDEX POSITION controls
These controls adjust the position of the index display on the monitor screen. The V control is for the vertical direction and the H is for the horizontal direction

⓫ SYNC IN connector (BNC connector)
Connect a time base corrector or external sync generator. The connector accepts composite sync signal.

⓬ CENTERING control
If a normal picture does not appear on the monitor screen in the playback mode, turn the control until you get the best possible picture. Normally set this control to the center detent position.

⓭ VIDEO OUT connector (BNC connector)
Connect to the video input of a video monitor or a time base corrector.

⓮ TV connector (8-pin connector)
Connect an 8-pin connector of a video monitor here.

⓯ EXT CPU ON/OFF switch
Normally set this switch to the OFF position.
Only when the player is to be operated by the external computers, remove the stopper, set the switch to ON and reinstall the stopper to keep the switch to the ON position. The any function and control keys on the control unit of the player will not function.
● When you want to operate the LDP-1000A with the external computers, please contact your Sony dealer.

⓰ SC/SYNC INT/EXT switch
When the player is to be operated synchronizing with the internal sync signal, set this switch to the INT position. When the player is to be operated synchronizing with the external sync signal connected to the SYNC IN (and SC IN) connector(s), set this switch to the EXT position. If no external sync signal is supplied to the SYNC IN connector, the player operates synchronizing with the internal sync signal regardless of the switch position.

⓱ EXT connector (RS232C serial interface connector)
Connect the external computers to operate the player.

⓲ LOCK PULSE jack
Use this jack to superimpose characters, pictures or graphic display from the Sony SMC-70 series microcomputer over the playback picture of the video disc. Connect this jack to the optional SMI-7073 RGB superimposer or SMI-7074 NTSC superimposer for the SMC-70 series using the remote cable supplied to the LDP-1000A to obtain a stable superimposition over a still or variable speed picture.

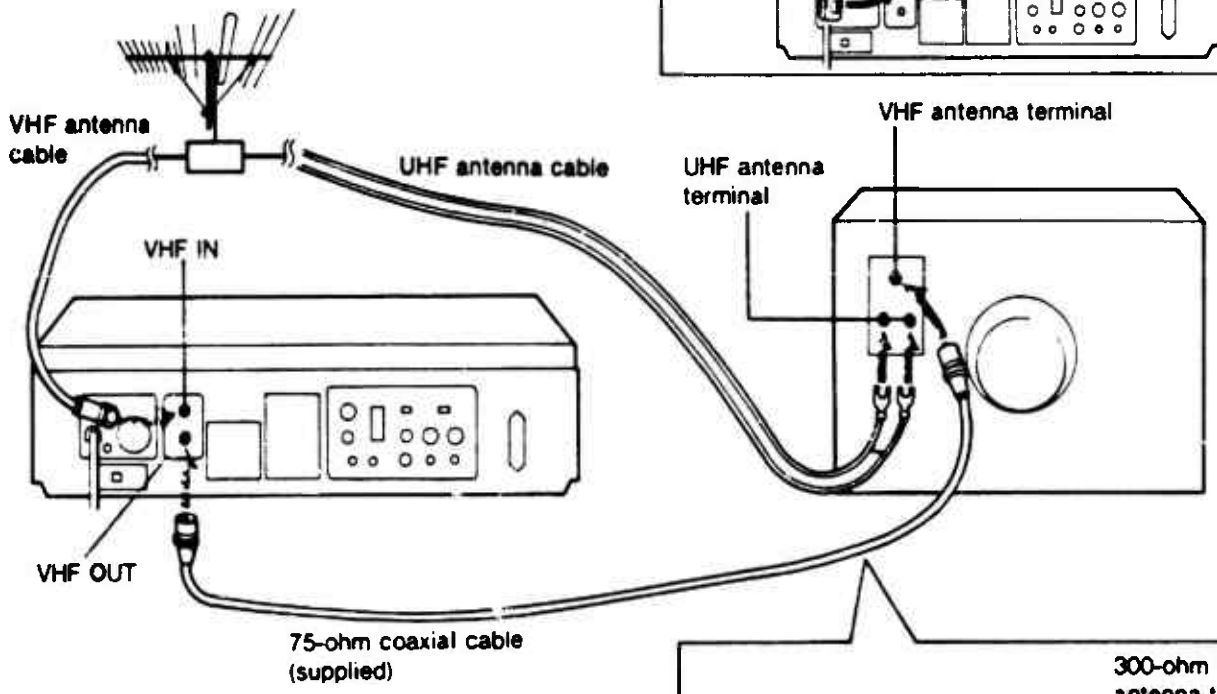
To see a picture, connect a TV receiver or a video monitor to the videodisc player as follows.
If you use a TV receiver, the TV receiver should be adjusted so that it displays a picture from the videodisc player.

TO SEE A PICTURE WITH A TV RECIEVER

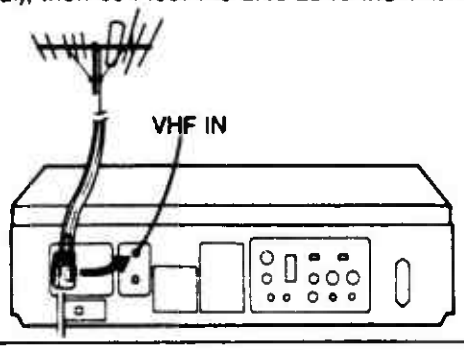
ANTENNA CONNECTION

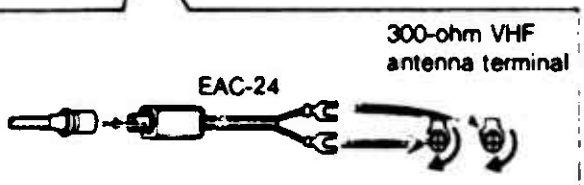
Remove the VHF antenna cable from the TV receiver and connect it to the player. Leave the UHF antenna cable connected to the TV receiver.

If your cable is a 75-ohm coaxial type (round) cable, connect it to the VHF IN with an F-type connector (optional).



If your cable is a 300-ohm ribbon type (flat) lead-in connect it to the EAC-25 external antenna connector (optional), then connect the EAC-25 to the VHF IN.





If your TV receiver is not equipped with an F-type VHF antenna terminal, connect the cable to the 300-ohm antenna terminals using the EAC-24 (optional).

TV CONNECTION

Once the VHF signal connection indicated above is completed, the VHF TV signals as well as the signal from the player can be fed to the TV receiver so that you can also view TV programs in the usual way.

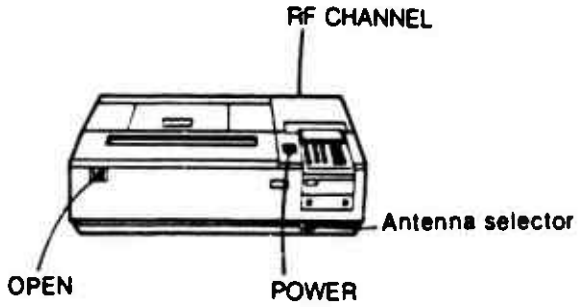
CAUTION
Connection between the LDP-1000A VHF OUT connector and the antenna terminals of a TV receiver should be made only as shown in these instructions.
Failure to do so may result in operation that violates the regulations of the Federal Communications Commission regarding the use and operation of rf devices. Never connect the output of the LDP-1000A to an antenna or make simultaneous (parallel) antenna and LDP-1000A connections at the antenna terminals of your receiver.

TV ADJUSTMENT

Adjust your TV receiver to eccept the signal from your pleyer in this way;

On the player

1. Set the RF CHANNEL selector loceted at the rear of the player to CH-3 or CH-4, whichever channel is not active in your erea.
2. Set the entenne selector to VDP.
3. Press the POWER button to turn on the pleyer.
4. Press the OPEN button and insert e disc. (See page 12.)



On the TV

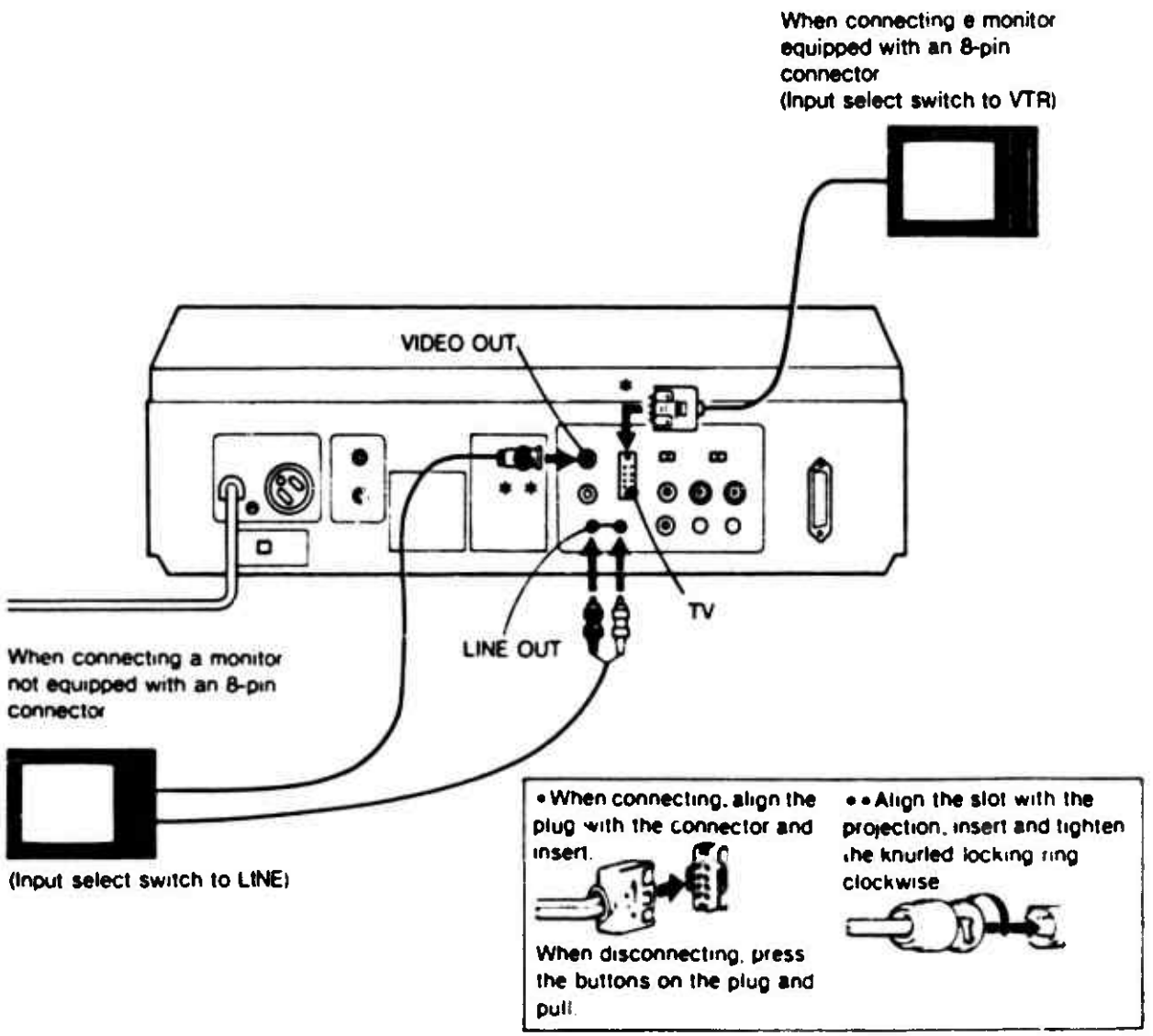
5. Turn on the TV.
6. Set the chennel on the TV receiver to the VHF channel 3 or 4, depending on the setting of the RF CHANNEL selector. The program of the disc will be displayed on the TV screen.
   If a picture does not appear on the TV screen or it the display is not clear, fine tune the channel on the TV.
   If your TV receiver has an electronic tuner and does not have a button for VHF channel 3 or 4, edjust e channel select button so thet the program of the disc is clearly displeyed on the TV screen end the sound is clearly heerd.

● For details about TV channel adjustment, see the instruction manuel turnished with the TV receiver.

Now the TV receiver has been correctly tuned to receive the signal trom the player.

TO SEE A PICTURE WITH A VIDEO MONITOR

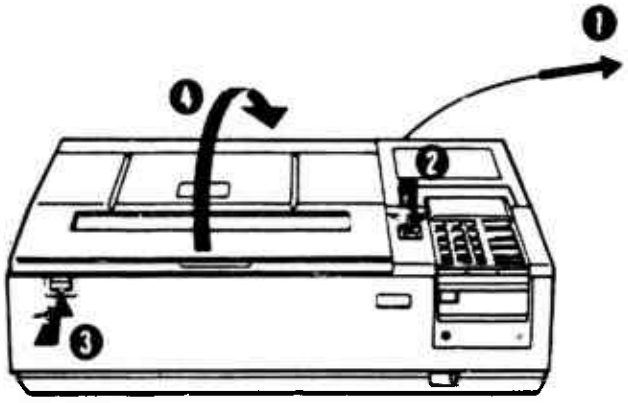
Once you connect a monitor, you can watch the pleyback picture on the monitor screen.



TO OPEN AND CLOSE THE DISC COMPARTMENT LID

To open

1. Plug the ac power cord into a wall outlet.
2. Press the POWER button.
3. Press the OPEN button.
   The lid will unlock and lift up slightly.
4. Lift up the lid all the way. Be careful not to force the lid beyond its normal open position.



To close

Push the lid down firmly so that the latch locks securely.

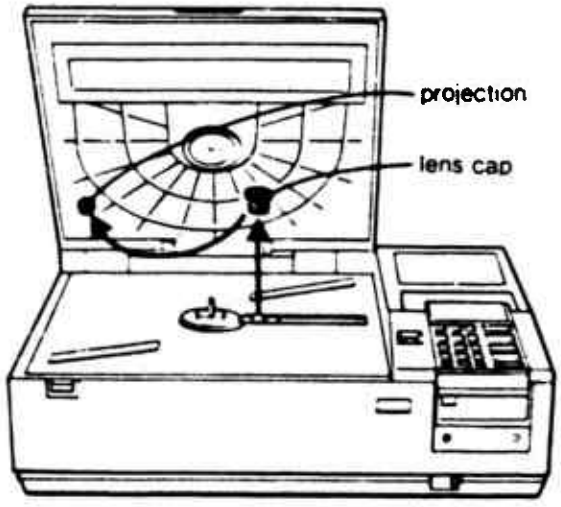
If the lid does not open

1. Turn the power off.
2. While pressing the OPEN button, press the POWER button again.

- If the lid still does not open, contact your Sony dealer.

REMOVE THE CAP OVER THE OBJECTIVE LENS

A cap has been put on the objective lens at the factory to protect the lens from damage and dust.
Open the disc compartment lid and remove the cap.

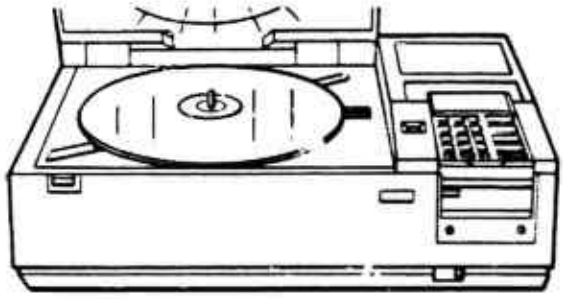


This cap should be saved for the later use when the player is shipped again or is not to be used for an extended period of time. Save the cap by putting it over the projection on the lid.

TO INSERT AND REMOVE THE VIDEO DISC

To Insert

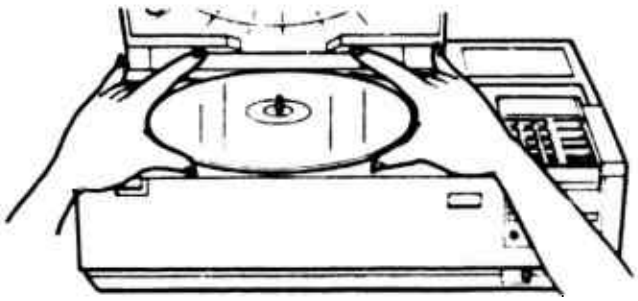
1. Open the disc compartment lid.
2. Place the disc with the desired program label up and install it to the center wheel firmly.



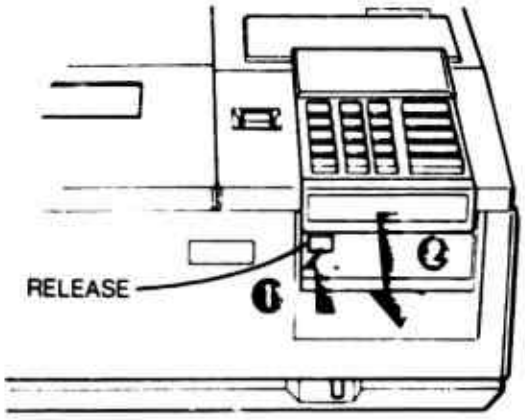
3. Close the lid.
   The disc will start rotating. The STANDBY lamp will blink for several seconds, then will go off to show that the player is ready to play.

To remove

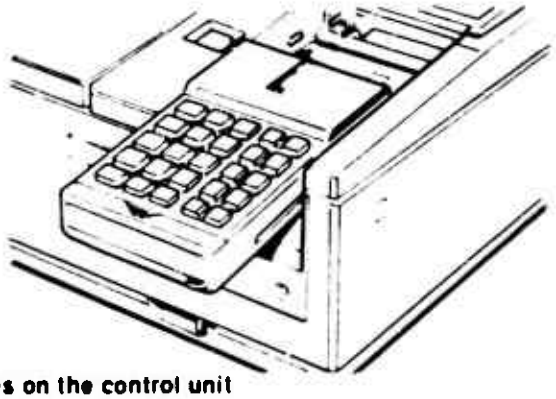
1. Press the OPEN button when you want to stop playing the disc, no matter what mode the player is in.
   The disc will stop rotating and the lid will unlock and lift up slightly.
2. Lift up the lid all the way.
3. Remove the disc, holding it by the rim.
4. Close the lid.



The control unit can be detached from the player by sliding it toward you while pressing the RELEASE button. It can be used either as a wireless or a wired remote control unit.



To attach, slide the control unit until it is plugged in firmly.

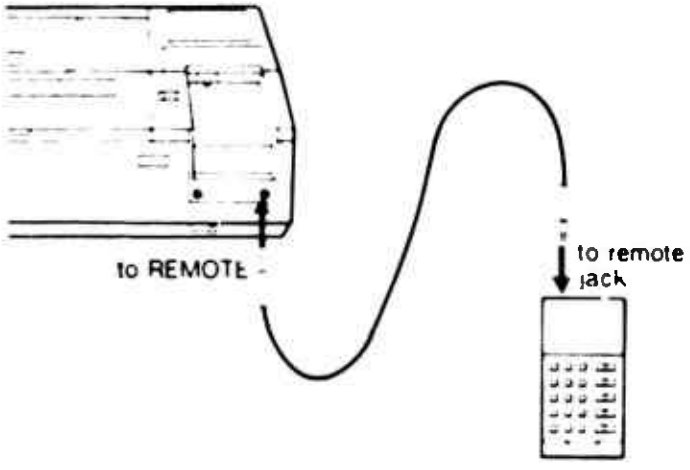


Notes on the control unit

- Keep the control unit away from hot or humid places.
- Avoid dropping any foreign objects into the control unit.
- To avoid a malfunction, do not press two or more function keys simultaneously.

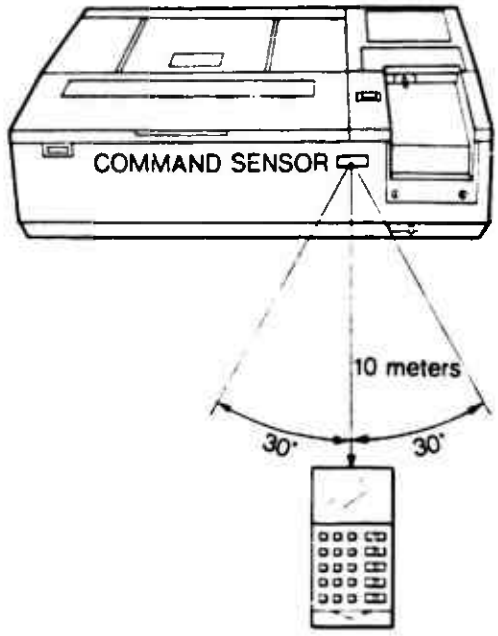
WIRED REMOTE CONTROL

Once the POWER button on the player is depressed and the player and the control unit are connected with the supplied remote cable, you can remotely control the player with the control unit detached from the player. Connect the remote cable to the remote jack on the control unit and the REMOTE jack on the player.



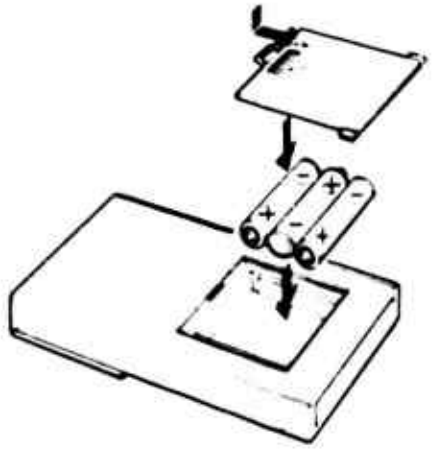
WIRELESS REMOTE CONTROL

Once the POWER button on the player is pressed, you can remotely control the player with the detached control unit up to 10 meters (3 feet) away from the player and within the range illustrated below.
The red lamp of the COMMAND SENSOR blinks to indicate that a function key has been pressed.



Battery installation

The control unit operates on batteries when it is used without the remote cable. To install the batteries, press down and slide open the battery compartment lid and install the three batteries size AA (IEC designation R6) with the correct polarity. Then close the lid.



- In normal operation, battery life is over six months. When the batteries are exhausted, the remote control unit will not operate the player properly. When this happens, replace all the batteries.
- If the control unit is not to be used for a long period of time, remove the batteries to avoid damage from possible battery leakage.

HOW TO PEAY A VIDEO DISC

OPERATION

There are three kinds of video discs; a CAV disc having the control instructions, a CAV disc having not the control instructions and a CLV disc. The operation is a little different among discs.

1. Press the POWER button.
2. Press the OPEN button and lift up the lid all the way.
3. Insert a video disc.
4. Close the lid. the STANDBY indicator lights and the disc starts rotating.
   - If you want to display the index on the monitor screen, press the INDEX key.

When a CAV disc having the control instructions is played, the PROGRAM and STEP/STILL indicators light 15 seconds after the lid is closed.

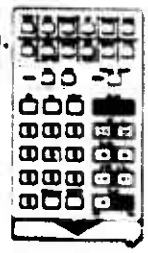
The menu will be displayed.
Press the keys according to the instructions of the displayed menu.

When a CAV disc having not the control instructions is played, the PLAY indicator lights 15 seconds after the lid is closed.

The playback will automatically begin.
You can use any function ksys on the control unit.
- For search operation, see page 16.

When a CLV disc is played, the CLV and PLAY indicators light 10 seconds after the lid is closed.

The playback will automatically begin.
- The [SCAN ◀◀] [SCAN ▶▶] and forward [PLAY ▶] keys can be used.
- For search operation, see page 16.



- To stop the playback momentarily, press the STOP key.
- To end the play of the disc, press the OPEN button, and the disc stops rotating and the lid is unlocked.

OPERATION WITH CONTROL INSTRUCTIONS

If "TOO LARGE" is displayed on the screen, → The number you have pressed is larger than the instructions on the menu. Wait until the display disappears for a few seconds, and press the correct keys.

To see the menu again, → Press the [MENU] key.

To interrupt the operation with control instructions momentsriiy,	→ Press the [INT] key. The playbeck stops end a still picture is obtained. You cen use eny function keys on the control unit.
To rsturn to ths operation with control inatructions interrupted by the [INT] key,	→ Press the [END] key, end the originel operetion will be resumed.
To atop ths oparation with the control instructiona,	→ Press the [CL] key or the [END] key. The pleybeck stops end you can press any function buttons.
To rsatart ths operation with control inatructions atopped by ths]CL] kay or [END] ksy,	→ Press the [MENU] key, or press the [PGM] key, then the [RUN] key, and the menu will be displeyed.
To skip to tha aegment to be playsd nsxt during the operation with tha control inaturctions,	→ Press the [SKIF] key.
If, while a aegmant is playing during the operation with control instructions, you dacida to aaa that asgmsnt from tha beginning oncs sgsin,	→ Press the [REVIEW] key.

TO SELECT THE AUDIO CHANNEL TO BE HEARD

Press the [AUDIO CH-1] or [CH-2] key. When the key is pressed, the sound is cut off, and when the key is pressed again, the sound can be heerd.

TO SKIP TO THE NEXT CHAPTER OR REVIEW THE CHAPTER BEING PLAYED (Chapter Stop)

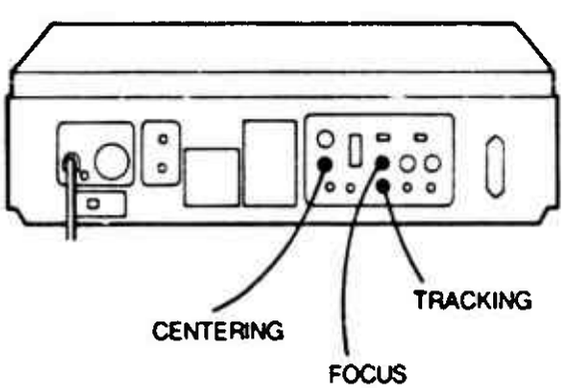
— For s disc with chaptsr data —

Press the [INDEX] key to displey the chepte number end keep the [SCAN] key depressed. (To skip to the next chapter press [SCAN ▶▶]; to return to the beginning of the chepter being played press [SCAN ◀◀].) When the beginning of the chapter is found, pleybeck stops and e still picture is obteined (CAV disc) or playbeck starts (CLV disc). Relesse the [SCAN] key at this point. The CAV disc will stert playback. To skip more chepters press [SCAN] egain.

IF A NORMAL PICTURE DOES NOT APPEAR ON THE MONITOR SCREEN

Turn the CENTERING control until you get the best possible picture. If the picture still conteins noise, adjust the FOCUS control and then the TRACKING control.

● After plsying this particular disc, return the FOCUS and TRACKING controls to their center detent position.

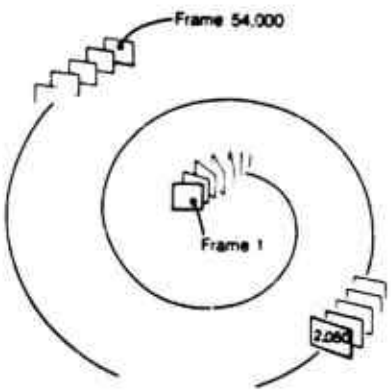


TO SEARCH FOR A PARTICULAR POINT (Point Search) (For CAV disc and CLV disc)

On a CLV disc, search for the desired point by frame.
On a CAV disc, search in reference to the elapsed playback time from the beginning of the disc (in minutes).

CAV disc: For example, to search for Frame 2050:

Step	Keys to be pressed	Display
1.	[SEARCH]	SEARCH 00000
2.	[2[[0[[5[[0[	SEARCH 02050
3.	[ENTER]	Frame 2050 will be displayed. (Still picture)



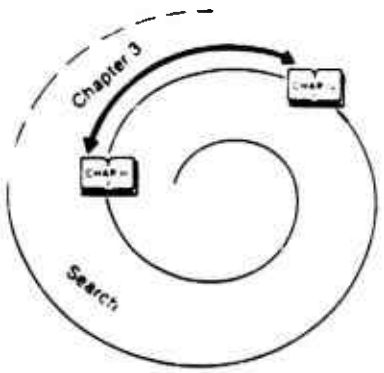
CLV disc: For example, to search for the 25-minute point:

Step	Keys to be pressed	Display
1.	[SEARCH]	SEARCH 00000
2.	[2[[5]	SEARCH 00025
3.	[ENTER[	Playback will begin from the 25-minute point.

TO SEARCH FOR THE BEGINNING OF A PARTICULAR CHAPTER (Chapter Search) (For CAV disc and CLV disc)

For example, to search for chapter 3:

Step	Keys to be pressed	Display
1.	[SEARCH]	SEARCH 00000
2.	[MODE]	SEARCH C-001
3.	[3[	SEARCH C-003
4.	[ENTER[	The beginning of Segment 3 will be searched for and the still picture of the first frame will be displayed (CAV disc), or playback starts (CLV disc).

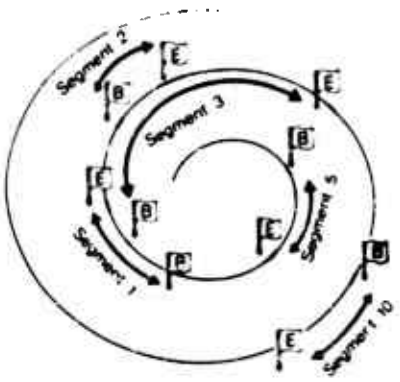


TO SEARCH FOR THE BEGINNING OF A PARTICULAR SEGMENT (Segment Search) (Only for the CAV disc)

To enter the segment data, refer to the "MEMORIZING THE SEGMENT DATA" on page 20.

For example, to search for Segment 10:

Step	Keys to be pressed	Display
1.	[SEARCH[	SEARCH 00000
2.	[MODE]	SEARCH C-001*
3.	[MODE[	SEARCH S-001
4.	[1[[0[* To search for Segment 1, skip this step	SEARCH S-010
5.	[ENTER[	The beginning of Segment 10 will be displayed.



*For a disc with no chapter data, "SEARCH S-001" will be displayed. In this case, skip Step 3.

TO REPEAT A PARTICULAR PART UP TO 15 TIMES (Point Repeat Search) (For CAV disc and CLV disc)

On a CAV disc, playback between any specified two frames at the specified speed can be done repeatedly up to 15 times.

CAV disc: For example, to play from Frame 100 to 170 three times at slow (x1/5) speed:

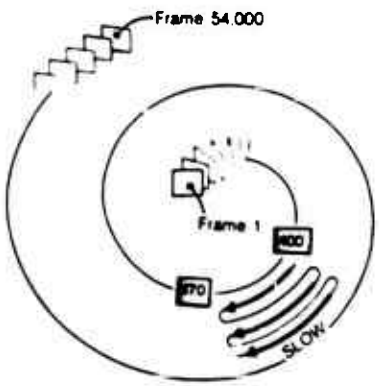
Step	Keys to be pressed	Display
1.	Search for Frame 100, the starting point.	
2.	[REPEAT]	00000 The frame number of the starting point is memorized.
3.	[1] [7] [0] (frame number of end point)	00170
4.	[SLOW] • To play at normal speed, skip this step.	SLOW 00170
5.	[ENTER]	REPEAT * 01
6.	[3]• (times to be repeated)	REPEAT * 03
7.	[ENTER]	The part from Frame 100 to Frame 170 will be played back at slow speed three times.

On a CLV disc, playback between any two points specified in minutes from the beginning can be done at normal speed repeatedly up to 15 times.

CLV disc: For example, to play from the 25-minute point to the 40-minute point three times:

Step	Keys to be pressed	Display
1.	Search for the 25-minute point, the starting point.	
2.	[REPEAT]	00000 The starting point in minutes is memorized.
3.	[040] (time of end point)	00040
4.	[ENTER]	REPEAT 01
5.	[3]• (times to be repeated)	REPEAT 03
6.	[ENTER]	The part from the 25-minute point to the 40-minute point will be played back three times. Playback will continue after the playback of this particular part ends.

• If [0] is pressed at this point, playback will be repeated until you press the [CL] key.



TO REPEAT A PARTICULAR CHAPTER UP TO 15 TIMES (Chapter Repeat Search) (For CAV disc and CLV disc)

For example, to repeat Chapter 12 twice:

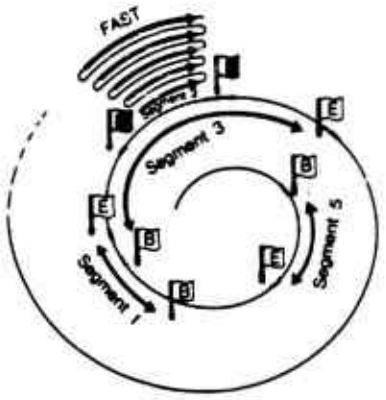
Step	Keys to be pressed	Display
1.	[REPEAT]	REPEAT 00000•
2.	[MODE]	REPEAT C-001
3.	[1] [2] (Chapter number)	REPEAT C-012
4.	[ENTER]	REPEAT * 01
5.	[2] (Times to be repeated)	REPEAT * 02
6.	[ENTER]	Chapter 12 will be played back twice. The CAV disc stops and the CLV disc plays back the next chapter.

• If "REPEAT C-001" is displayed in step 1, skip step 2.

TO REPEAT A PARTICULAR SEGMENT UP TO 15 TIMES (Segment Repeat Search) (Only for the CAV disc)

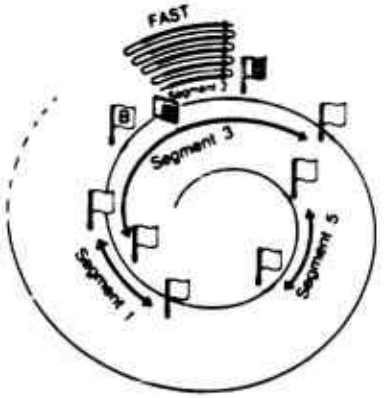
For example, to repeat Segment 2 for five times at fast (×3) speed:

Step	Kaya to be praaaed	Displtay
1.	Search for a segment to be played.	
2.	[REPEAT]	REPEAT S-001
3.	[2] (segment number) • For Segment 1, skip this step.	REPEAT S-002
4.	[FAST] • To play at normal speed, skip this step.	FAST S-002
5.	[ENTER]	REPEAT ✕ 01
6.	[5] (times to be repeated)	REPEAT ✕ 05
7.	[ENTER]	Segment 2 will be played back five times at fast speed.



You can atart the Segment Rapeat Search trom any point.

Stap	Keya to be praaaad	Display
1.	Search for the frame to start playback.	
2.	[REPEAT]	00000
3.	[MODE]	REPEAT S-001 •
4.	[2]	REPEAT S-002
5.	[FAST]	FAST S-002
6.	[ENTER]	REPEAT ✕ 01
7.	[5]	REPEAT ✕ 05
6.	[ENTER]	

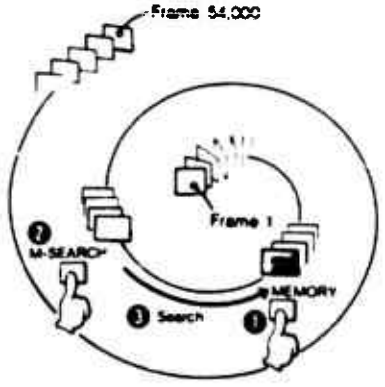


•If the disc has chapter data, "REPEAT C-001" will be displayed. In this case press [MODE] again.

TO SEARCH FOR A FRAME WHICH HAS BEEN MEMORIZED IN ADVANCE (Memory Search) (Only for CAV disc)

1. Play a video disc in any mode.
2. Press the [MEMORY] key at the point you want to see again.
3. Press the [M. SEARCH] key. The point where the [MEMORY] key was pressed will be searched for.

● If the [MEMORY] key has been pressed several times, only the point where the key was last pressed will be searched for.

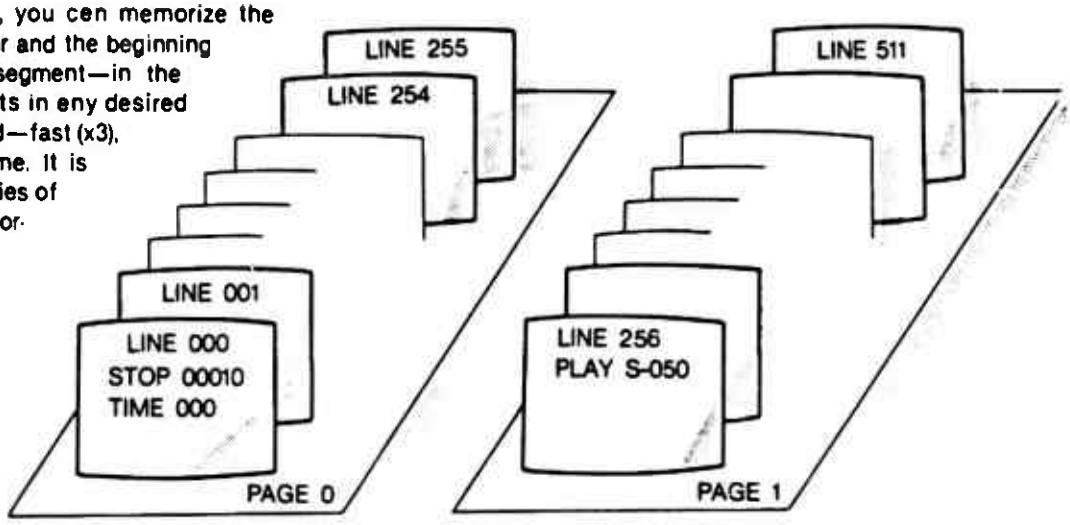


TO PLAY AT PARTICULAR SLOW SPEED (Only for the CAV disc)

You can select a playback speed from 1/1 to 1/255 normal speed by entering the denominator of the desired speed as data for the playback speed. For example, to play at speed 1/30 normal speed, "30" should be pressed:

1. Press the [STEP/STILL ▶‖] key for forward play or the [STEP/STILL ‖◀] key for reverse play.
2. Press the [3] and [0] keys (data for the playback speed).
3. Press the [ENTER] key. Playback at a speed 1/30 normal speed will begin.

Using the built-in microprocessor, you can memorize the segment data—the segment number and the beginning and end frame numbers of the segment—in the LDP-1000A and play these segments in any desired order and also at any playback speed—fast (x3), normal, slow (x1/5) or frame-by-frame. It is also possible to jump to another series of playback program during the performance of a series of playback program. This is called "Branching". The control instructions for playing or branching are stored on numbers 000 to 511 in the memory. These numbers are called "Line Numbers".



On some CAV discs, the segment data and control instructions are recorded on the beginning portion of the audio channel 2. You can use these data to play the disc, of course, and also play the same disc with another control instructions when you enter the new data by using the control unit of the LDP-1000A.

This part of instruction manual tells you how to memorize the segment data and the control instructions for playing and branching.

MEMORY OF THE DATA TO BE ENTERED

The memory of the entered segment data and control instructions will be kept on the LDP-1000A for three days, or until new data or control instruction is entered.

MEMORIZING THE SEGMENT DATA

PREPARATION

1. Make the necessary connections.
2. Insert a video disc.
3. Press the [INDEX] key to display the frame number.
4. Play the video disc and note the frame number of the beginning and end frame of each segment on the monitor screen.

OPERATION

If, for example, you want to memorize Segment 5 whose beginning frame number is 123 and end frame number is 500, procaes as follows:

Step	Keys to be pressed	Display
1.	[SEG]	SEGMENT * 001
2.	[5] (segment number)	SEGMENT * 005
	• To memorize the data for segment 1, skip this step.	
3.	[ENTER]	005 00000] 00000 (] blinks; 005 = segment number)
4.	[1] [2] [3] (beginning frame number)	005 00123] 00000
5.	[ENTER]	005 00123 00000]
6.	[5] [0] [0] (end frame number)	005 00123 00500]
7.	[ENTER]	SEGMENT * 006 (006 = number of the next segment)
8.	[CL]	

• To memorize the following segments, repeat steps 3 through 7.

TO CHECK OR TO CORRECT THE SEGMENT DATA

1. Press the [SEG] key. "SEGMENT * 001" will be displayed.
2. Enter the segment number to be checked or corrected.
   • To check the data of Segment 1, skip this step.
3. Press the [ENTER] key and "005 00123] 00500" will appear.
   (005 = segment number; 00123 = beginning of the segment; 00500 = end of the segment)
   • **To correct the beginning frame number,** press the [CE] key to clear the entered data and enter the correct data.
4. Press the [ENTER] key again. The display is changed to "005 00123 00500]".
   • **To correct the end frame number,** press the [CE] key to clear the entered data and enter the correct data.
5. Press the [ENTER] key and the data of the next segment will appear.

• To finish the check or correction of the segment data, press the [CL] key.

Segment data sheet

Segment number	Beginning frame number	End frame number
Seg 1	800	1200
Seg 2	1550	1700
Seg 3	1000	3500
Seg 4	3015	3700
Seg 5	123	500
Seg 6	....	....
Seg 7	....	....

ON THE MARK "]"
When the index displayed on the screen includes two data, the] mark blinks. You can enter the data here.

005 00123] 00500
(] blinks. (You can enter the data here.))

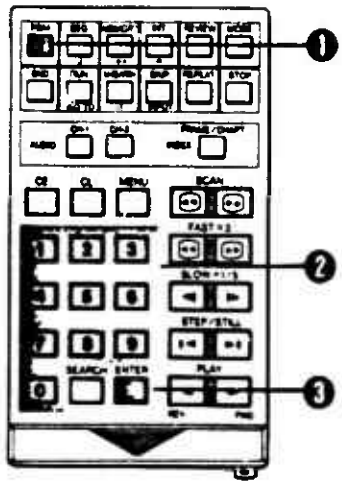
PREPARATION

1. Make the necessary connections.
2. Press the POWER button.
3. Insert a video disc.
4. Press the INDEX key to display the frame number.

TO START ENTERING CONTROL INSTRUCTIONS

1. Press the [PGM] key. "START AT 000" will be displayed.
2. Press the Line Number to which you want to assign a control Instruction. To start from Line 000, skip this step.
3. Press the [ENTER] key. "000 FUNCTION?" will be displayed.

Now the LDP-1000A is ready to memorize the data for control instructions for playing or for branching.



Note:
Because all data is automatically erased from a Line when new data is entered, we recommend checking that there is no data you want to retain stored on a Line before you enter new data on that Line.

TO END ENTERING CONTROL INSTRUCTIONS

When you have finished memorizing the all control instructions for a program, terminate the program with the following method.

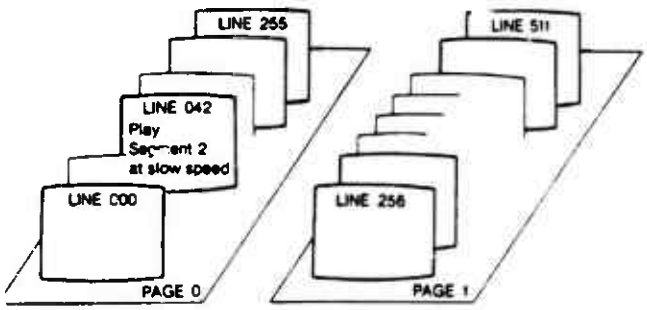
Press the [ENTER] key while "000 FUNCTION?" is displayed. "000 END 00000" will appear.
line number, present frame number

MEMORIZING THE CONTROL INSTRUCTIONS FOR PLAYING

TO PLAY SEGMENTS AT FAST (x3), NORMAL OR SLOW (x1/5) SPEEDS

For example, if you want to enter the control instructions "Play Segment 2 at slow (x1/5) speed" on Line 042, proceed as follows:

Step	Keys to be pressed	
	[PGM] [4] [2] [ENTER]	042 FUNCTION? (line number)
1.	[SLOW]	042 SLOW S-001 (segment number to be played; playback speed)
	• For playback at normal speed, skip this step.	
2.	[2] (segment number) • To play Segment 1,	042 SLOW S-002
3.	[ENTER]	044 FUNCTION?



- For the following control instructions, repeat these steps.
- To end entering control instructions, press the [ENTER] key again.

TO SELECT AUDIO CHANNEL 1 OR 2

Both audio channels will be played back in normal operation. However, you can select the audio of either channel 1 or 2 alone in this way:

1. Display the "000 FUNCTION?".
2. Press the [CH-1] key for channel 1 or the [CH-2] key for channel 2.
   "000 AUDIO-1 0" or "000 AUDIO-2 0" will be displayed.
3. Press one of the following keys:
   [0] for off (the sound will be cut off)
   [1] for on (the sound will be heard)
   [2] for toggle (the mode—on or off—will be changed)
4. Press the [ENTER] key.

TO DECIDE WHETHER TO DISPLAY THE INDEX OR NOT

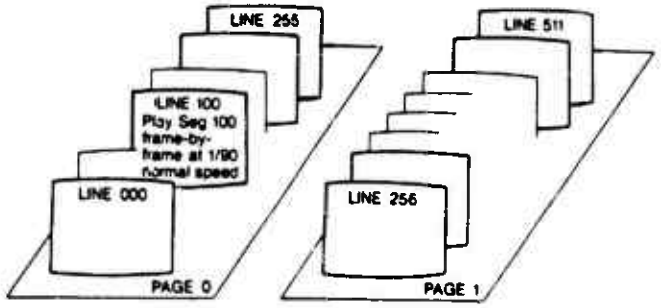
You can decide whether the Index is displayed or not during playing a segment in this way:

1. Display the "000 FUNCTION?".
2. Press the [INDEX] key and the "000 INDEX 0" will be displayed.
3. Press one of the following keys:
   [0] for off (the index is not displayed)
   [1] for on (the index is displayed)
   [2] for toggle (the mode—the index is displayed or not—will be changed)
4. Press the [ENTER] key.

TO PLAY A SEGMENT FRAME-BY-FRAME

The speed at which frames are to be advanced can range from 1/1 to 1/255 normel speed. Enter the denominetor of the desired playback speed as data.
For exemple, if you went to enter the control Instructions "Play Segment 55 frame-by-frame at 1/90 normal speed" on Line 100, proceed as follows:

Step	Keys to be praaaed	Display
	[PGM] [1] [0] [0][ENTER]	100 FUNCTION? (line number)
1.	[STEP]	100 STEP S-001
2.	[5] [5] (segment number) • To play Segment 1, skip this step.	100 STEP S-055
3.	[ENTER]	100 STEP 000
4.	[9] [0] (denominetor of the playback speed)	100 STEP 090
5.	[ENTER]	103 FUNCTION?

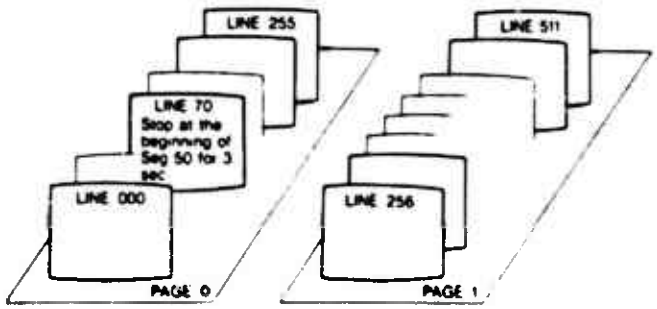


- For additional segments, repeat these steps.
- To end entering control instructions, press the [ENTER] key again.

TO OBTAIN A STILL PICTURE OF THE BEGINNING OF A PARTICULAR SEGMENT

For example, if you want to enter the control instructions "Stop at the beginning of Segment 50 for 3 seconds" on Line 70, proceed es follows.

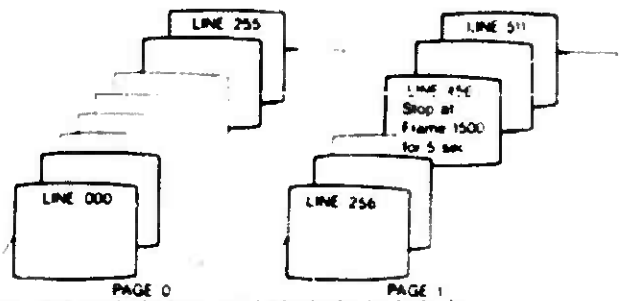
Stap	Kaya to be prassed	Display
	[PGM] [7] [0] [ENTER]	070 FUNCTION? (line number)
1.	[STOP]	070 STOP S-001
2.	[5] [0] (segment number) • For Segment 1, skip this step.	070 STOP S-050
3.	[ENTER]	070 TIME 000
4.	[3] (numbe: of seconds the frame is to be displayed)	070 TIME 003
5.	[ENTER]	073 FUNCTION?



- To end entering control instructions, press the [ENTER] key again.

- You can display a specific frame by designating by the frame number es follows. For example, to enter the control instructions "Stop at Frame 1500 for 5 seconds" on Line 456:

Step	Kaya to be pressed	Display
	[PGM] [4] [5] [6] [ENTER]	456 FUNCTION?
1.	[STOP]	456 STOP S-001
2.	[MODE]	456 STOP 00000
3.	[1] [5] [0] [0]	456 STOP 01500
4.	[ENTER]	456 TIME 000
5.	[5]	456 TIME 005
6.	[ENTER]	460 FUNCTION?

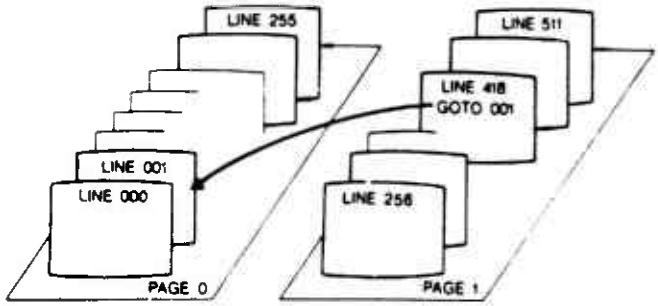


MEMORIZING THE CONTROL INSTRUCTIONS FOR BRANCHING

USING THE [GO TO] KEY

You can jump from one line to another.
For example, if you want to enter control instructions "Jump to Line 001" on Line 418, proceed as follows.

Step	Keys to be pressed	Display
	[PGM] [4] [1] [8] [ENTER]	418 FUNCTION?
1.	[GO TO (RUN)]	418 GO TO 000
2.	[1] (line number to jump to) • If jumping to Line 000 skip this step.	418 GO TO 001
3.	[ENTER]	420 FUNCTION?



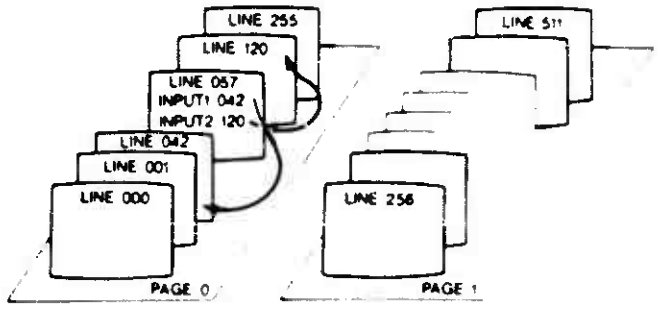
• To end entering control instructions, press the [ENTER] key again.

• You can jump to any line among Line 000 to Line 511.

USING THE [INPUT] KEY

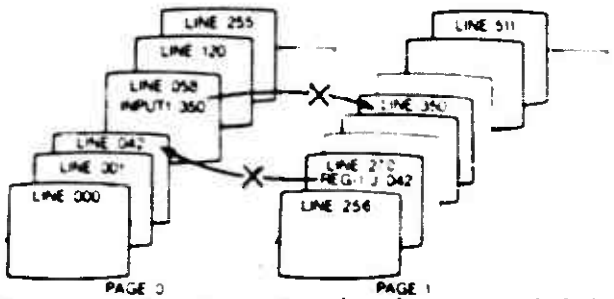
Use the [INPUT (SKIP)] key to devise a control instruction which offers the user a choice of which one of up to 9 different lines will follow.
It is possible, for example, to devise a Line 057, in which if the user presses figure key 1 the next control instruction will be Line 042 and if he presses figure key 2 the next will be Line 120, and so on. Proceed in this way:

Step	Keys to be pressed	Display
	[PGM] [5] [7] [ENTER]	057 FUNCTION?
1.	[INPUT]	057 INPUT-1 000
2.	[4] [2] (line number to jump to)	057 INPUT-1 042
3.	[ENTER]	057 INPUT-2 000
4.	[1] [2] [0]	057 INPUT-2 120
5.	[ENTER]	057 INPUT-3 000
6.	[ENTER]	060 FUNCTION?



• To end entering control instructions, press the [ENTER] key. "000 FUNCTION?" will be displayed.

NOTE
As lines are stored in two groups (which we call "Pages") — from Line 000 to Line 255 on Page 0 and from Line 256 to Line 511 on Page 1, it is not possible to jump from a line on one page to a line on another.



USING THE BUILT-IN REGISTERS

The LDP-1000A has four registers numbered from 0 to 3. The registers count the times a control instruction hes been performed. The velue memorized on e register decreases by one eech time the control instruction is performed. If the velue on e register is not 0, the operation returns to the beginning of that control instructions to repeet it until the velue becomes 0, at which point the operation goes on to the next control instructions. Using these registers, you cen repeat a particular control instructions up to 255 times.

NOTE
Again, it is not possible to jump from a line on one page to a line on another.

For example, if you want to enter control instructions "Play Segment 5 at fast (x3) speed for three times using Register-1" on Line 040, proceed as follows:

Step	Keys to be preaaed	Display
	[PGM] [4] [0] [ENTER]	040 FUNCTION?
1.	[= (INT)]	040 REG-0] = 000
2.	[1] (register number) ● To use Register-0, skip this step.	040 REG-1] = 000
3.	[ENTER]	040 REG-1 = 000]
4.	[3]* (times to be played)	040 REG-1 = 003]
5.	[ENTER]	042 FUNCTION?
6.	[FAST] ● To play at normal speed, skip this step	042 FAST S-001
7.	[5] (segment number) ● To play Segment 1, skip this step.	042 FAST S-005
8.	[ENTER]	044 FUNCTION?
9.	[J]	044 REG-0] J 000
10.	[1] (the same register number as in step 2)	044 REG-1] J 000
11.	[ENTER]	044 REG-1 J 000]
12.	[4] [2] (line number in step 5) ● If the line number is 000, skip this step	044 REG-1 J 042]
13.	[ENTER]	046 FUNCTION

*If "0" is entered at this point, the register does not function.

● To end entering control instructions, press the [ENTER] key again.

TO CHECK THE CONTROL INSTRUCTIONS ENTERED

1. Press the [PGM] key. "START AT 000" will be displayed.
2. Press the line number to be checked. To check Line 000, skip this step.
3. Press the [PGM] key. The date on the selected line will be displayed.

● To check the following control instructions, press the [PGM] key again.
● To end the check, press the [CL] key.

TO CORRECT THE ENTERED CONTROL INSTRUCTION DATA

Call up the line to be corrected and enter the correct data. For example, It you want to change the playback speed of Segment 2 stored on Line 006 trom PLAY to SLOW, proceed as follows:

Step	Keys to be preaaed	Display
1.	[PGM]	START AT 000
2.	[6] (line number)	START AT 006
3.	[ENTER]	006 FUNCTION?
4.	[SLOW]	006 SLOW S-001
5.	[2] (segment number)	006 SLOW S-002
6.	[ENTER]	006 FUNCTION?
7.	[END]	

● To end the correction, be sure to press the [END] key. If you press the [ENTER] key instead, the control instructions will be terminated at Line 006 and the next control instructions will not be performed.

ERROR MESSAG

If you press an improper key, one of the following messeges will appeer on the monitor screen for about 1.5 seconds. When this happens, press the correct key.

INVALID KEY
"INVALID KEY" indicetes thet the pressed key hes no function in the operation of the LDP-1000A.

TOO LARGE
"TOO LARGE" eppeers in the tollowing situations.

- when a repeat time of more then 16 is entered in the Repeat Search mode.
  (The maximum number of repeet times is 15.)
- when e register number of 4 or more is entered.
  (The register numbers are trom 0 to 3.)
- when you select Line 512 or higher to enter date.
  (The lines are numbered from 000 to 511.)
- when a segment number of 64 or higher is entered.
  (The segments are numbered from 1 to 63.)
- when 256 or higher is entered as data for the register.
  (The register cen accept data trom 1 to 255.)

FORMAT ERROR
When you press a key which designates an impossible operetion while checking the control instructions or performing the operation with control instructions "FORMAT ERROR" will appear and operation will be interrupted.

PAGING ERROR
When you enter the control instructions in the LDP-1000A to jump to a Line stored on the other page using the [INPUT] key or [J] key, "PAGING ERROR" will appear. Remember that Lines 000 to 255 are stored on one page and Lines 256 to 511 on the other page and that you cannot jump between pages.

MEMORY END
When you enter control instructions over Program 511 during entering control instructions, "MEMORY END" will appear.

CAPACITY OF DATA

Frame numbers: trom 1 to 54,000
Segment numbers: trom 1 to 63
Line numbers: trom 000 to 511 (Line 000 to 255 are stored on Page 0 and Line 256 to 511 are stored on Page 1.)
Inputs: trom 1 to 9
Register numbers: trom 0 to 3
Values which can be entered on a register: trom 1 to 255
(It 0 is entered, the register does not function.)
Times a particular part or segment can be repeated: up to 15
(It 0 is entered, playback will continue until the command to stop is entered.)
Speed tor frame-by frame advance or slow motion: 1/1 to 1/255

TO START THE OPERATION WITH CONTROL INSTRUCTIONS

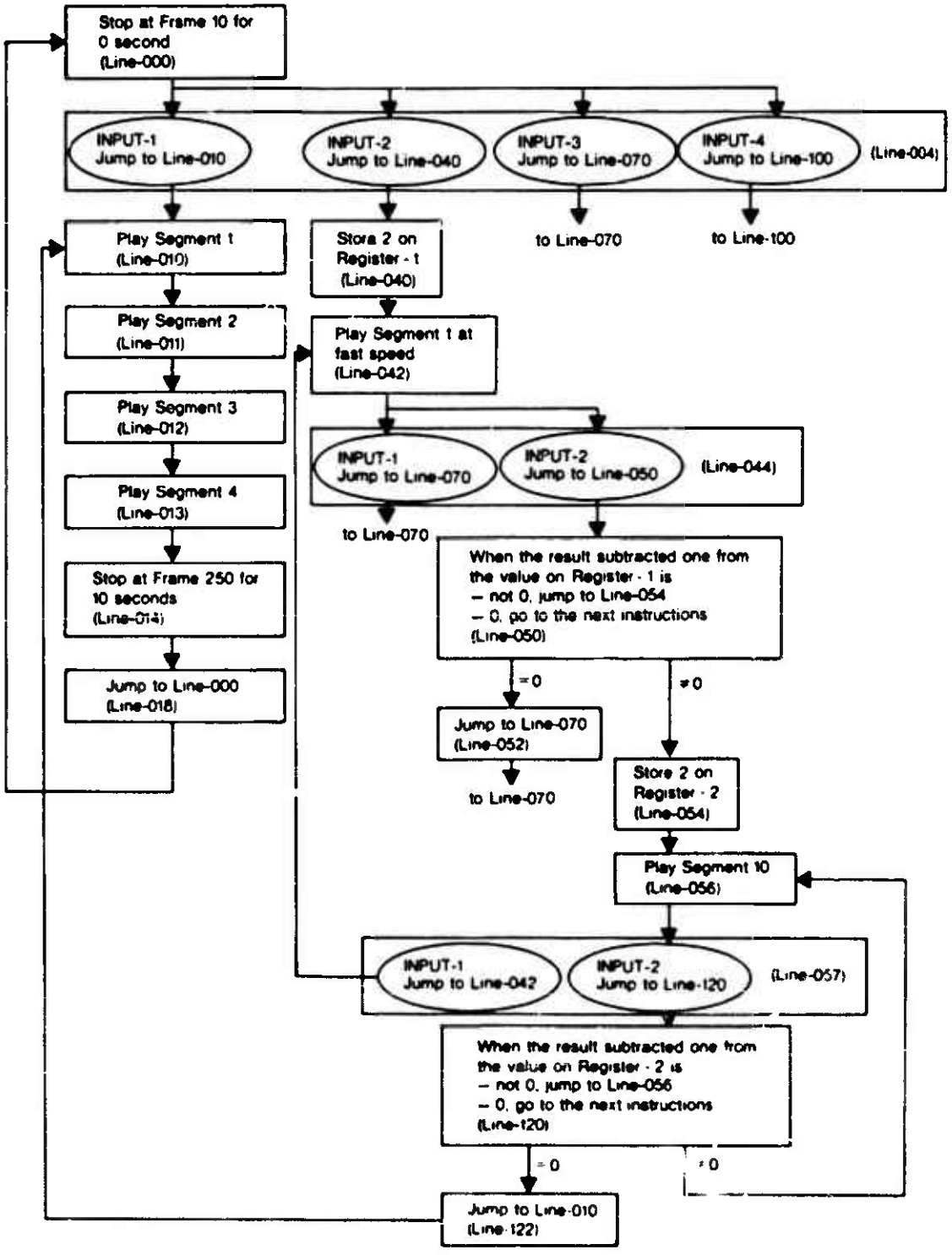
1. Press the [PGM] key.
2. Press the figure keys of the line number you went to pley.
3. Press the [RUN] key, and the operation will begin end continue in sequence to the line on which the end indication is entered.

EXAMPLE OF CONTROL INSTRUCTIONS

CONTROL INSTRUCTION SHEET

Line number	Instruction	Meaning of the instruction
000	STOP 00010 TIME 000	Stop et Frame 10 tor 0 second.
004	INPUT-1 010 INPUT-2 040 INPUT-3 070 INPUT-4 100	Jump to Line 010 when [1] is pressed. Jump to Line 040 when [2] is pressed. Jump to Line 070 when [3] is pressed. Jump to Line 100 when [4] is pressed.
009	END	End of the instructions.
010	S-001	Play Segment 1.
011	S-002	Pley Segmenl 2.
012	S-003	Play Segment 3.
013	S-004	Play Segment 4.
014	STOP 00250 TIME 010	Stop at Frame 250 tor 10 seconds.
018	GO TO 000	Jump to Line 000
020	END	End ot the instructions.
040	REG-1 002	"2" is stored on Register-1.
042	FAST S-005	Play Segment 5 at tast (x3) speed.
044	INPUT-1 070 INPUT-2 050	Jump to Line 070 when [1] is pressed. Jump to Line 050 when [2] is pressed.
047	END	End ot the instructions.
050	REG-1 J 054	When the result subtracted one trom the value on Register-1 is not 0, jump tc Line 054; when it is 0, go to the next instructions.
052	GO TO 070	Jump to Line 070
054	REG-2 002	"2" is stored on Register-2.
056	S-010	Play Segment 10.
057	INPUT-1 042 INPUT-2 120	Jump to Line 042 when [1] is pressed. Jump to Line 120 when [2] is pressed.
060	END	End ot the instructions.
120	REG-2 J 056	When the result subtracted one trom the value on Register-2 is not 0, jump to Line 056, when it is 0, go to the next instructions.
122	GO TO 010	Jump to Line 010

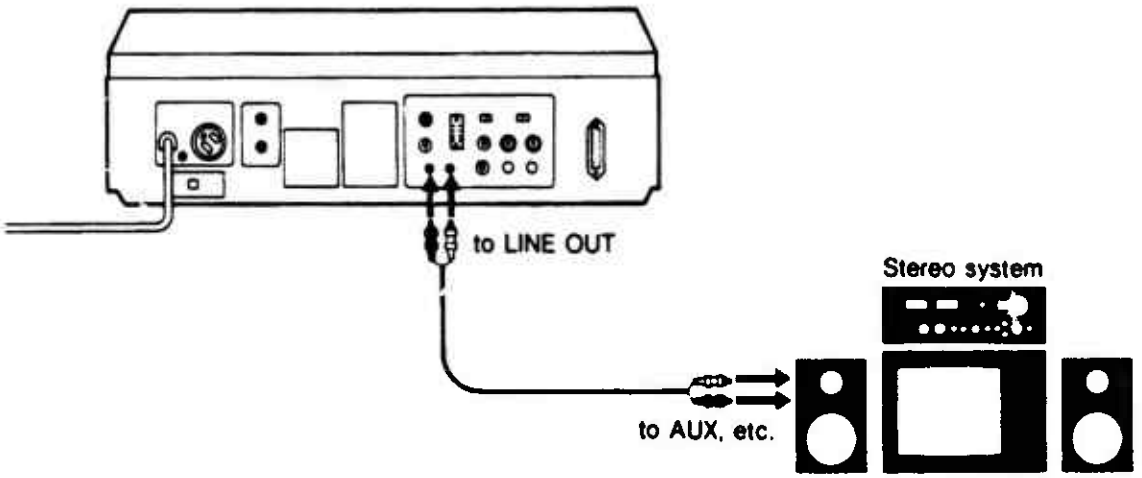
HOW THE CONTROL INSTRUCTIONS ON THE PREVIOUS PAGE OPERATES



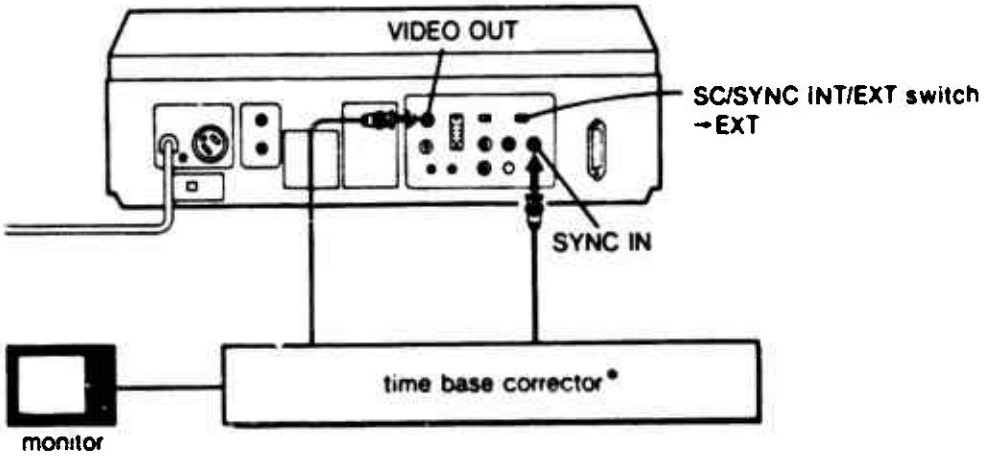
OPTIONAL CONNECTIONS

STEREO SYSTEM CONNECTION

Stereo hi-fi bilingual sound can be reproduced by connecting the stereo system to the LDP-1000A videodisc player.
Connect the LINE OUT CH-1/L and CH-2/R on the player to the AUX input (or tape tuner input) jacks on the stereo amplifier or receiver.

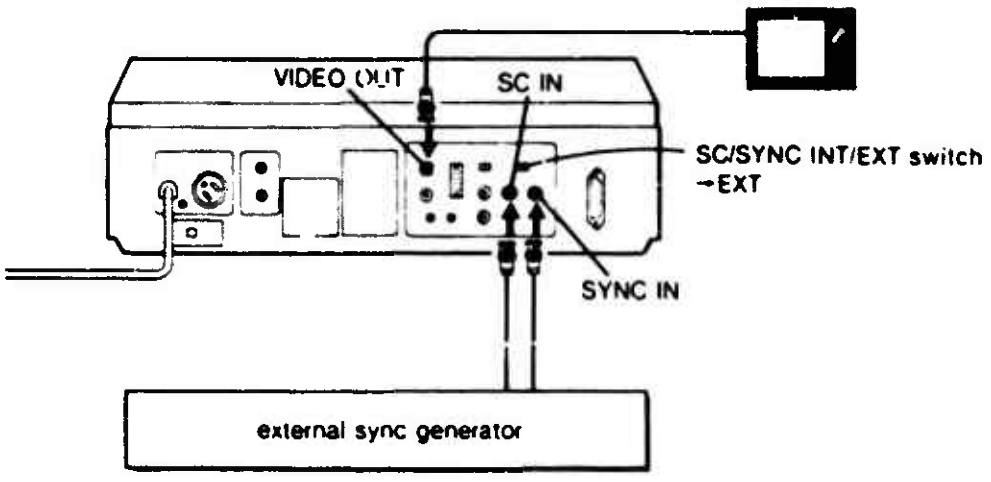


CONNECTION WITH A TIME BASE CORRECTOR



● If a monitor is connected to the LDP-1000A, the color reproduction cannot be obtained on the monitor screen.
• The time base corrector should be the direct mode type such as the Sony BVT-1000, BVT-2000.

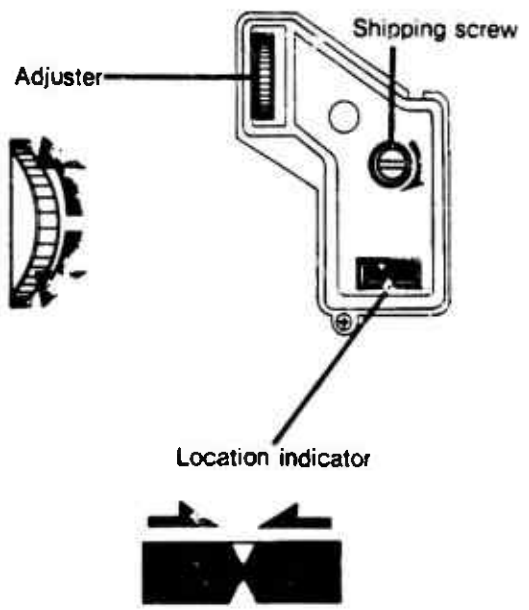
CONNECTION WITH AN EXTERNAL SYNC GENERATOR



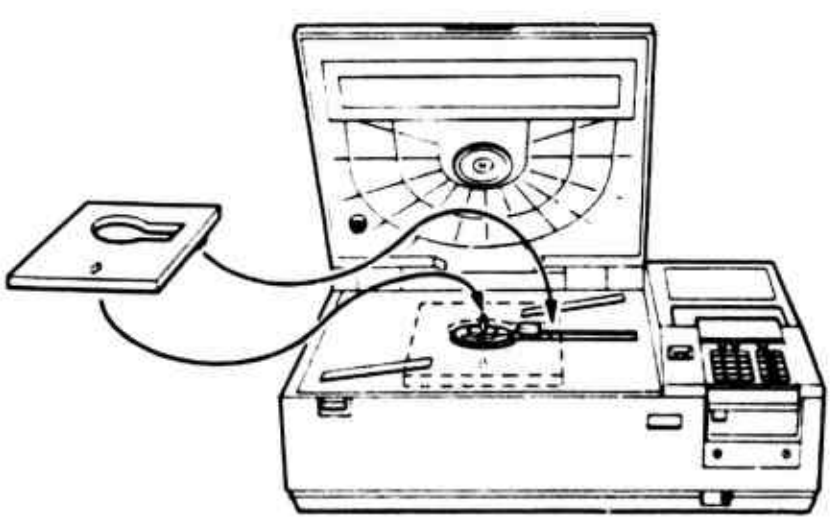
● If the external sync generator is not connected to the SC IN connector, the color reproduction cannot be obtained on the monitor screen.

TO TRANSPORT THE LDP-1000A

1. Place the cap over the objective lens.
2. Turn the adjustor on the bottom until the arrow of the location indicetor are eligned.
3. Turn the shipping screw clockwise with a coin or similar object until it is tight.

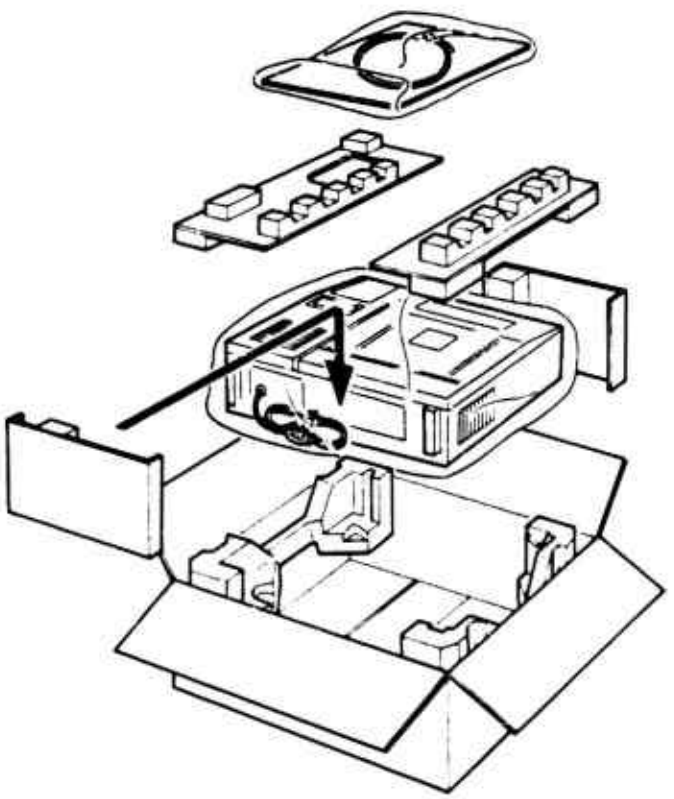


4. Attach the cushion in the disc compartment as illustrated.



REPACKING FOR SHIPMENT

The packing procedure is subject to change.
Refer to the illustretion on the originel carton for later pecking instructions.



SPECIFICATIONS

General

Pick-up method	Laser beam (reflective)
Laser	He-Ne (λ = 8328 Å)
Maximum playing time	
	CAV: 30 min./side
	CLV: 60 min./side
Spindle revolution	CAV: 1800 r.p.m.
	CLV: 1800 r.p.m. (inner circumference) to 700 r.p.m. (outer circumference)

Video

Signal	EIA standards, NTSC color
Output	1.0 V(p-p), 75 ohms unbalanced, sync negative
Resolution	Color: 360 lines
Signal-to-noise ratio	42 dB
VHF output	Channel 3 or 4 (selectable) 75 ohms, unbalanced
Input signals	SYNC 4 V(p-p) ±1 V(p-p), 75 ohms SC 2 V(p-p) ±0.5 V(p-p), 75 ohms

Audio

Output	LINE OUT: Less than 2 k ohms 0 ±2 dB [V] (100% MOD, 47 k ohm load), unbalanced PHONES: 8 ohms, −19 ±2 dB [V]
Signal-to-noise ratio	More than 50 dB
Frequency response	40 Hz to 20 kHz (±3 dB)

Others

Power requirements	120 V ac ±10%, 60 Hz
AC OUT	Unswitched 120 V ac, max. 400W
Power consumption	110W
Operating temperature	10°C to 35°C (40°F to 95°F)
Dimensions	542 × 160 × 415 mm (w/h/d) (21 3/8 × 6 1/4 × 16 3/8 inches)
Weight	19.6 kg (43 lb 3 oz)

Remote control unit RM-1002

Remote control system	
	Infrared control
Power requirements	4.5 V dc Battery size AA × 3 (IEC battery designation R6)
Dimensions	91 × 31 × 157 mm (w/h/d) (3 5/8 × 1 1/4 × 6 1/4 inches)
Weight	280 g (10 oz) including batteries

Accessories supplied

75-ohm coaxial cable with F-type connectors (1.5m)
Remote cable (2m)

Design and specifications subject to change without notice.

TROUBLE CHECKS

Meny epparent melfunctions mey be caused by e mis-set control thet has been overlooked, or some other equelly simple ceuse.
Should eny difficulty erise in operation, check through this list of symptoms, end cause end remedy. Should the difficulty persist, unplug the unit end contect the deeler from whom this unit wes purchased, or e locel Sony euthorized service stetion.

Symptom	Cause and remedy
Lid does not opan.	● Power cord is not proparly plugged into ec outlet. → Connect en ec power cord. ● Power is not turned on. → Press the POWER button. ● Latch is not relaased. → Press the OPEN button. (See page 12.)
Disc does not rotate.	● Power is not turned on. → Connect the ec power cord to en ec outlet end press the POWER button. ● Lid is not shut completely. → Push lid close firmly so letch locks.
Disc rotates but no picture.	● The shipping screw has not been loosened. → Turn the shipping screw fully counter-clockwise. ● The lens cap is still put on. → Remove the lens cap. ● The unrecorded side is pleyed. → Turn over the disc. ● TV or monitor is not turned on. → Turn the TV or monitor on. ● Wrong connection from player to TV or monitor. → Meke correct connections. ● TV set is not tuned to channal 3 or 4. → Set TV to chennel 3 or 4 (inective channel in your eree). ● The RF CHANNEL selector is not set proparly. → Set the selector to the inactive channel in your erea. ● The player is in the stendby mode. → Weit for e moment. ● The EXT CPU ON/OFF switch is set to ON. → Set tha switch to OFF.
Picture quality is bad.	● Bed connections between pleyer end TV or monitor → Check all connections, particulery F typa connector. ● The RF CHANNEL selector setting end the channel selected on the TV are not correspond. → Both TV end pleye: must be set to the same channel (3 or 4) which is not ective in your area. ● TV fine tuning has not been edjusted. → Fine tune the TV for optimum picture quality. ● The disc is not instelled firmly. → Instell the disc firmly. ● Poor disc. → Try playing e different disc. If other discs give good quelity, the problem is with the particular disc.
TV no longer receives the broadcasting program efter it hes been connected to the pleyer.	● The entenna ceble has not been connected. → Connect the VHF entenna cable properly. ● The antenne selector on the pleyer is set to VDP. → Set the selector to ANT.
A particuler part of e particular disc is not reproduced properly.	● The disc is demeged. → Press the SCAN key to skip over the dameged portion.
The control unit on the pleyer does not work.	● The unit has not been connected firmly. → Install the unit onto the pleyer firmly.
Wireless remote control does not work.	● The batteries in the control unit ere exhausted. → Replece the batteries.
Wirad remote control does not work.	● Remote control ceble is not connected firmly. → Connect the cable to the REMOTE jack on the pleyer end the remote control jack on the control unit firmly.

APPENDIX D

LENCO PSG-310 MANUAL

PSG–310

SYNC GENERATOR

INSTRUCTION MANUAL

TABLE OF CONTENTS

GENERAL DESCRIPTION

The PSG-310 Digital Color Sync Generator exemplifies the latest in design techniques of digital engineering. The unique circuits allow us to offer an ultra-stable and trouble free generator, with exclusive features not normally found in broadcast quality equipment.

A temperature compensated crystal oscillator, operating at 14.318180 MHz provides the stable master frequency source from which all pulses and subcarrier are derived. Using digital dividing techniques, subcarrier, as well as the sync, blanking, horizontal and vertical drive pulses, are produced virtually jitter free. All pulse widths and levels are fixed per EIA standards and cannot change. There are only three internal adjustments in the generator — subcarrier amplitude, lock range and vertical phase — which are set at the factory and seldom, if ever, require adjustment.

The Genlock circuit has a unique noise immunity circuit which makes the generator highly insensitive to noise or extreme changes in input levels. The incoming video is sensed by an extremely fast video presence detector, processed and locks the generator automatically to the 50% point of the sync pulse. Genlock is accomplished within one second.

Other exclusive features include a clamped video feed, a field ident pulse is available if burst flag is not required, continuous adjustment of horizontal phase, instead of in 70 nS steps, vertical phase adjustable to two lines advance to compensate for enhancers, and three voltage regulators with current overload devices for power buss and module protection. The PSG-310 supplies a full set of drive signals to both the output connectors on the rear panel as well as to the busses on the frame for use by any other 300 System module that may require them.

SPECIFICATIONS

PULSE:

Outputs	One each: Sync, Blanking, H-Drive, V-Drive & Burst Fleg or Field Ident
Levels	4.0 Volts ±5% p-p, fixed
Impedance	75 Ohms, ±1%, source terminated
Widths	Nominal FCC, EIA and NTSC standards
Tilt and Overshoot	Less than 1%
Jitter	Less then 5nS, referenced to subcarrier
Rise Time	120 nS ± 20 nS

SUBCARRIER:

Frequency	3.579545 MHz, ±5 Hz
Output Impedance	75 Ohms, ±1%, source termineted
Output Level	2.0 volts p-p, edjusteble ± 20%
Harmonic Distortion	Greater then −45 dB et 2.0 volts p-p

GENLOCK MODE:

Video Input Level	1.0 volt p-p, ±6 dB, RS-170/NTSC signel
Video Input Impedance	Greeter than 50K bridging, looping input
Pulse Jitter	Less than 5 nS, referenced to input sync
Subcarrier Jitter	Less than 0.5° referenced to input burst

FRONT PANEL CONTROLS:

Crystal Frequency	±30 Hz et 3.579545 MHz
Subcarrier Phase	360° Continuous
Horizontal Phase	0.8 μS delay to 3.5 μS advence, continuous

INTERNAL CONTROLS:

Subcarrier Amplitude	±6 dB from 2.0 volts p-p
Vertical Phase	1.5 lines edvance to 1.5 lines delay in ½ line steps

ENVIRONMENTAL:

Temperature	0° to 50° C, embient

POWER:

Input	+15 (145 mA), −15 (130 mA) and +8 (120 mA) VDC received from PFM-300 Freme

MECHANICAL:

Size	2⅞" (7.30 cm) x 1½" (3.81 cm) x 11⅞" (30.163 cm)
Weight	11.15 oz (316 g)
Mounting	PFM-300 System frame only (one module width)

SET-UP PROCEDURE

EQUIPMENT REQUIRED:

PFM-300 System Frame and Power Supply
Dual Trace Oscilloscope with 2-75 Ohm BNC through, Terminations
Frequency Counter 10 MHz
PEX-308 Extender Board

PROCEDURE:

1. With Generator in Frame, apply power.
2. Using a short 75 Ohm Cable and a feed through 75 Ohm Termination, check for 4 Vp-p ± .5V of each output signal.
   Output #3 — Burst Flag or Field I.D.
   Output #4 — Composite SYNC
   Output #5 — Composite Blanking
   Output #6 — Vertical Drive
   Output #7 — Horizontal Drive
   Output #8 — Subcarrier (should be 2Vp-p)
3. If Subcarrier Amplitude adjustment is not required, go to step 4. If required, place on Extender and reconnect the subcarrier output to the Oscilloscope as in 2. Adjust L4 for 2 Vp-p.
4. Connect the subcarrier output to a frequency counter. Check for a 3.579545 MHz ±5 Hz frequency. Adjust the front panel frequency control if required.
   NOTE: This adjustment affects the internal reference only and should not be adjusted for any Genlock funtion.
5. Loop A Video source through BNC Terminals 1 and 2 and Terminate at the Oscilloscope, Channel 2. Connect Channel 1 to output #4 (Composite SYNC). Trigger Oscilloscope on Channel 1 and set sweep to 10 μS/Div. Set display to ALT and adjust gains for viewing the Video and SYNC.
6. Place the Generator in Genlock. The regenerated SYNC should lock to the Video SYNC. Adjust C40 for Lock. Apply a test signal to the input of the PSG-310 and observe the error voltage on R67. R67 is a stand up resistor. Connect the oscilloscope probe to the "loop" on R67 and then adjust **C40, LOCK FREQUENCY**, so that the error voltage is 0.5 volts above ground when the unit is cold. This voltage may have to be slightly readjusted if the unit does not lock properly.
7. Connect A 75 ohm Cable between output #6 (Vertical Drive) and the External Trigger input on oscilloscope. Switch oscilloscope to Ext. Trigger. Check that the regenerated vertical SYNC pulses align with Video SYNC. Adjust R62 if required.

TECHNICAL DESCRIPTION

This description is divided into ten sections (Figure 1). They consist of pulse generator, pulse output, subcarrier generator, subcarrier output, video processing, color lock, horizontal timing, vertical timing, video sense and power supply.

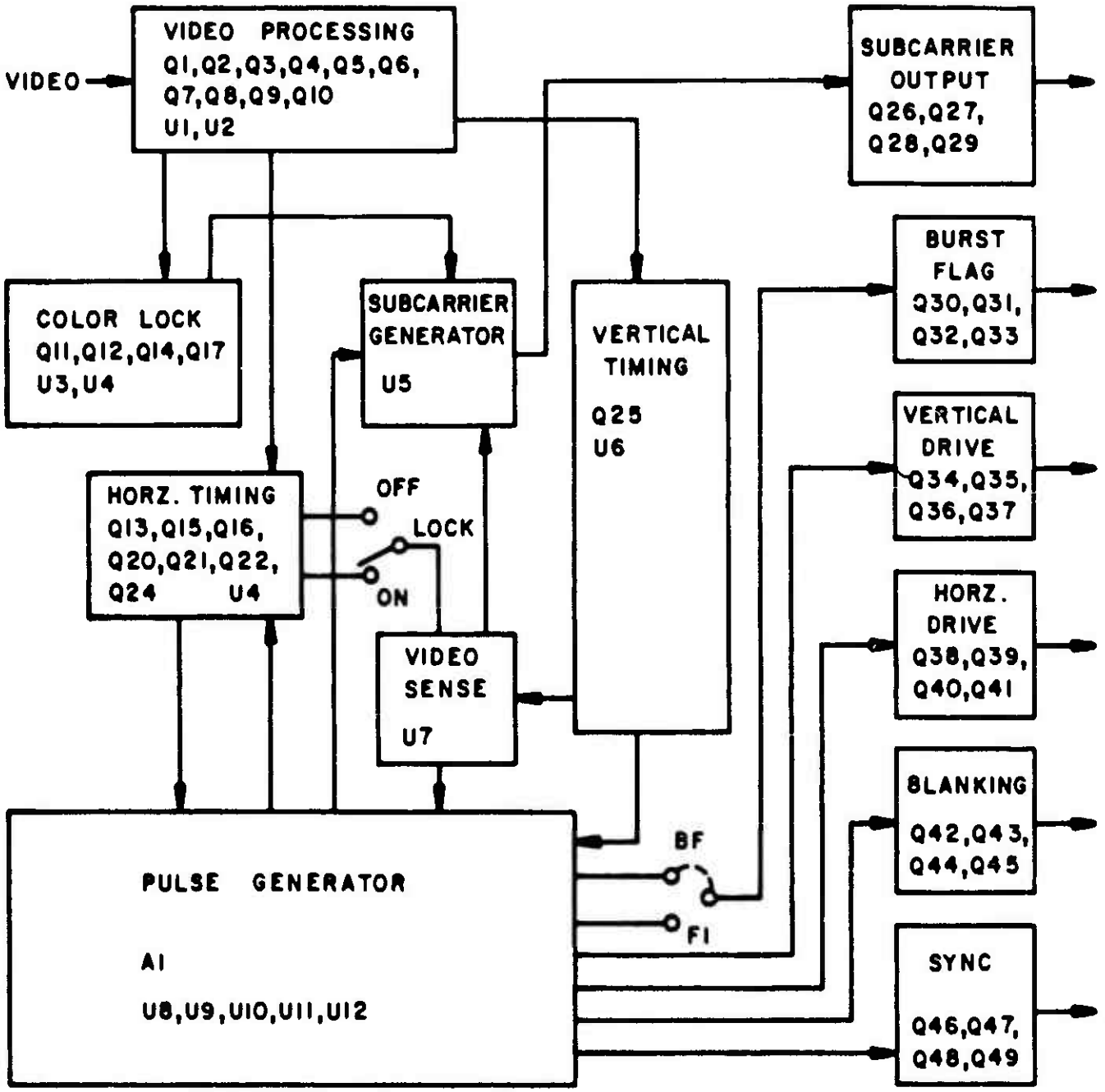


FIGURE 1

PULSE GENERATOR

The internal frequency reference is a self-contained 14.31818 MHz oscilletor (A1). A1 is switched on and off with Q18 and Q19. The oscillator is turned off whenever the Genlock Mode of operation is selected. Its output is coupled to Z8 through R68 and C38. Z8 is e gate controlled two-channel-input wideband amplifier. The output on pin 7 is coupled back into an input through crystal Y2. This positive feedback causes the amplifier to oscillate at 14.3 MHz. When the oscillator A1 is operating, it causes the VCO Z8 to sympathetically lock. When A1 is off, D12 (a varicap diode) controls the frequency while C40 sets the center frequency. D13 and D14 form a decoupled power source for Z8; D13 and D14 regulete et ±6.8 volts.

The buffered output (from Q23) drives 6 J-K flip-flops, Z9, Z10 and Z11. Z9 is wired as a synchronous divide by 4, its output (pin 12) is 3.579545 MHz. Z10 and Z11 are wired as a synchronous divide by 7, the output (Z11, pin 8) is 2.045 MHz. R89 and C49 limit the rise and fall times to meet the drive requirements of the sync generator IC Z12. The clock pulse is on pin 3.

Z12 is a completely self-contained sync generetor. It develops all the timing pulses prescribed by RS-170. The outputs used in this generator are: pin 16 composite sync, pin 15 horizontal drive, pin 14 composite blanking, pin 12 burst tlag, pin 11 vertical drive, and pin 9 field ident. Pin 5 is a vertical reset input. A negative going pulse on pin 5 will cause the generetor to reset to line number 11. All the outputs have the same amplitude, +3.5 to −8 volts.

PULSE OUTPUTS

All of the output cells ere the same. The sync output is typical. The output from Z12, pin 16, is coupled into the cell by R96. D33 end D34 clip the 11.5 Vp-p pulse to 4 Vp-p. D33 reverses at −3.4 volts and limits the negative swing to that level. D34 reverses at 0.6 volts end limits the positive swing to that level. The output cell has a gain of 2 so that the output et the emitters of Q48 and Q49 is 8 Vp-p.

The output cell is a high gain op-amp, with negative teedback provided by R124. R124 and R123 set the gain at 2. C60 limits the frequency response ot the cell and prevents parasitic oscillations. R126 sets the output impedance to 75 ohms.

SUBCARRIER GENERATOR

The subcarrier generator (Z5) is a gate controlled two-channel-input, wideband amplifier. When pin 2 is high, the 3.58 signal from Z9 is amplified. When pin 2 is low, the output from pin 7 is coupled back into the input through crystal Y1. This positive feedback causes the amplitier to oscillate at 3.58 MHz. D7 (e varicap diode) controls the frequency. D8 and D9 form a decoupled power source tor Z5. D8 end D9 regulate et ±6 8 volts.

SUBCARRIER OUTPUT

Q26, Q27, Q28 and Q29 make up an output cell identical to the PULSE OUTPUT cell with the exception of the input 3.58 tilter, L4 and C53. This tilter is a high Q resonant circuit tuned to 3.58 MHz. Its orimary function is to remove any harmonics in the incoming 3.58 MHz square wave signal. Its secondary function is to operate as a subcarrier emplitude control. This is done by de-tuning the circuit to decrease the signal at the input to the cell. C55 isolates the DC component from the output and R101 sets the output impedance at 75 ohms.

VIDEO PROCESSING

Input video is decoupled by C1 end C2 into FET Q1. Q1, Q2, Q3 and Q4 form an op-amp with R4 providing the negative teedback to set the input gain at 2. Biasing tor the amplifier is provided by op-amp Z1. This op-amp is the teedback amplitier tor the "feedback" clamp used in the input processing.

The clamped video at the emitters of Q3 and Q4 is bandpass tiltered by R10, L1 and C9 (approximately 500 KHz) Q7 is a sync pre-stripper; it makes a very coarse strip. The stripped sync from Q7 controls the gate ot Q9, a "sample and hold" circuit. The sample is taken at video sync tip and the tip voltage is stored in C13.

Z1, e voltaga follower, prevents loading of C13. R23 end R24 divide tha voltaga output of Z1 by 2/3, forming a refarence for the comparator Z2. The reference is always 2/3 sync amplitude. This insures that the fine stripper (Z2) elways strips ef tha 66% point of incoming sync. The stripped sync is on pin 7 of Z2.

Q10 is e "box car" one-shot. It davelops a 2 μS pulse following eech sync pulse. This pulse controls the gate of Q8, a "sample and hold" circuit. The sampla is made at beck porch end stored in C12. R16 bypasses e smell emount of the video around tha sampla and hold circuit. This ellows the clamp to "start up" at turn on.

Q5 end Q6 form e noise cancellin, circuit. Non-filtared video is coupled into Q5 through C7. Q5 is biased slightly off so that only impulse noise more negetive than sync tip will cause it to conduct. When it conducts, the collector will go positive, turning on Q6. C10 has little affect on Q6's turn on time, but delays the turn off time ebout 2 μS. The video going to tha sync stripper is low pass filtered to about 500 KHz which delays it ebout 1 μS. The strip reference voltaqe et Z2 pin 2 is pulled down eway from the noise by Q6 before tha noise arrives et pin 3 (tha noise is defected from non-delayed video). Q6 does not releasa the reference voltage until efter the impulse has anded (the turn off delay due to C10). In this way the noisa is not stripped out by Z2; this in turn helps control the jitter in the regenerated pulses.

COLOR LOCK

The color lock consists of e closed loop made up of a chroma demodulator, a chroma clamp, a "sample and hold" circuit, a faadback amplifier, and a subcarriar oscilletor. Z3, a balanced chrome demoduletor, is driven by two input signals. The first is chroma from bandpass filter L2 and C17. The second is from the subcarrier output amplifier through R52 and C32. Z3 contains e voltage controlled phase shifting circuit, with the input et pin 3, tha output at pin 1, end the control voltage at pin 2. A $\pm 45°$ range is provided with R29, the subcarrier control. The subcarrier et pin 1 is coupled to the demodulator section through the quadrature network C27, L3 end R38 to pin 6, the B-Y damodulator and pin 12, the R-Y demodulator. The R-Y (pin 10) and the B-Y (pin 8) outputs from Z3 ara switched with S1 to couple either one to the phase comparator. Since the burst information conteined in these two signals is 90° different in phase, the reganerated subcarrier locked to burst will be 90° displaced, depending on which one is selected by S1.

The chroma input to Z3 is either pin 14 or pin 15, depending on whether Q11 or Q12 is shunting tha chroma signal to ground through C19 or C20. The demodulated output will change 180° as the input is switched etween pin 14 and pin 15. In this wey as S2 directs the signal to either pin 14 or pin 15, the regenerated subcarrier will change 180°.

With these three controls, S1, S2 end R29, the regenerated subcarrier can be phased 360° with respect to incoming video burst phase. S1 controls it in 90° steps, while S2 controls it in 180° steps and R29 phases it linearly 90°. The selected output from S1 is coupled to the clamp Q14. Q14 clamps by shunting the signal on C18 to ground during sync tip. The clamp dnva is the prastnp pulse from Q7. Q17, e "sample and hold" circuit semples the clemped demodulated chroma during burst. The sample dnve pulse is from Q10. The vertical drive signal from Z12 (pin 11) prevents samples during vertical interval, when burst is omitted through D5. The sampled voltege is stored in C30. Op-amp Z4 is tha feedback emplifier end maintains a control voltage on D7 in the 3.58 oscillator circuit.

HORIZONTAL TIMING

Horizontal timing consists of a closed loop made up of a sample pulse generetor, a ramp generator, e feedback amplifier, and a 14.3 MHz oscillator. The sample pulses ara derived from stripped sync (Z2 pin 7). Z6 is a 31.5 KHz "lock out". It is a 40 μS one-shot triggered by the leading edge of sync. The output drives a "box car" one shot (Q15) that generates the sample pulse.

Q16 is in serias with Q15 and form a logical AND gate. The pulse on the basa of Q16 is in time with the ramp signal and gates the samole pulse out except during the ramp. This prevents erroneous samples that might occur due to noise on incoming video. The output from the AND gate Q15 and Q16, is coupled to Q13 where the "box car" one-shot develops the sample pulse.

The ramp signal is derived from horizontal drive. The drive pulse from Q40 and Q41 is level shifted by R90 and R84 to drive the one-shot Z7. The duration of the output pulse (pin 13) is determined by C45, R82 and R55, the HORIZONTAL PHASE control. The "box car" one-shot Q24 generates a 2 μS pulse at the trailing edge of the pulse form Z7. This in tum drives the AND gate Q15, Q16 and the ramp generetor Q21 (a Miller integrator). A remp forms at the collector ot Q21. Q20 is a "sample and hold" circuit where the sample pulse from Q13 samples the ramp from Q21 and stores the sampled voltage in C39. The feedback amplifier Z4 maintains a control voltage on D12 to control the trequency of the 14.3 MHz oscilletor.

VERTICAL TIMING

Vertical timing is achieved by resetting the generator iC (Z12) during the vertical interval. The reset pulse is generated by Q25 and drives pin 5 ot Z12. The timing pulse is derived from stripped sync (Z2, pin 7). C52 and R63 separate out the vertical pulses by integration. The one-shot Z6 is triggered neer the leading edge of the first vertical pulse. The duration ot the output pulse (Z6 pin 4) is controlled by C50, R61 end R62, the VERT PHASE control. C48 couples this pulse into the "box car" one-shot Q25 where the reset pulse is generated et its trailing edge.

VIDEO SENSE

The vertical reset pulse from Z6, pin 13 is used to trigger e one-shot, Z7. The pulse duretion is set by C46 and R83 to last several vertical fremes. Z7 is e retriggereble one-shot and will remain triggered so long as vertical resets ere detected. It video is removed from the generator, vertical resets will no longer be generated and Z7 will be ellowed to finish its pulse. At that time the genereter will automatically revert to internel mode.

The generator is teken in and out of lock by the video sense circuit through S3. When S3 is closed to Z7 pin 12, it will genlock it pin 12 is high (video present). It will revert to internal mode it pin 12 is low (no video). It pin S3 is open to pin 12, the generator will operate in internal mode. When S3 is in the genlock OFF position, the junction of R50 and R51 is held et a constant DC level (Z7 pin 12). This prevents eny signal at the output of Z4 from disturbing the subcarrier being generated in Z5.

POWER SUPPLY

The generator operates on a $\pm$15V source end a +8 volt source. On-board regulators Z13 (+5), Z14 (+12) and Z15 (−12) regulete these supplies. The generator will operate so long as the input supplies ere: +15V to +25V, −15V to −25V, end +8V to +15V.

Z14, Z15 and Z16 are all over temperature and over current protected.

MOTHER BOARD CONNECTOR PIN ASSIGNMENT

Provides interconnects between the modules and the power supply, as well as between modules to allow for maximum flexibility. The following table will allow for quick reference to what signals are available:

Mother Board Connector Aa Seen From Front:

	Pin	Pin	
+8 volt D.C. supply	1	A	System Ground
Input from BNC #1 or #2	2	B	Shield of BNC connector #1 & 2
		C	Input from BNC #1 or #2
Composite Reference Signal	8	J	+15 volt D.C. supply
		K	Burst flag or field ident
Sync	10	L	Blanking
Horizontal Drive	11	M	Vertical Drive
Output BNC #4	12	N	Output BNC #3
Output BNC #6	13	P	Output BNC #5
Output BNC #8	14	R	Output BNC #7
−15 volt D.C. supply	15	S	Subcarrier

PSG-310 SYNC GENERATOR
REAR PANEL CONNECTORS

1. Video in
2. Video In
3. Burst Flag or Field Ident. Output
4. Sync Output
5. Blanking Output
6. Vertical Drive
7. Horizontal Drive Output
8. Subcarrier Output

PSG-310 Parts List

Symbol	Description	Lenco Part#
A1	14.31818 MHz Oscillator OE-10	4700010
C1	1 μF 35V Tant T368A105K035AS	2105102
C2	1μF 35V Tant T368A105K035AS	2105102
C3	1μF 35V Tant T368A105K035AS	2105102
C4	1μF 35V Tant T368A105K035AS	2105102
C5	10pF Mica CM04CD100D03	2100100
C6	1μF 35V Tant T368A105K035AS	2105102
C7	250pF Mica CM05FD251J03	2101250
C8	.01μF 50V Ceramic UK-50-103	2103102
C9	.001μF Mica CM05FD102J03	2102100
C10	.001μF Mica CM05FD102J03	2102100
C11	330pF Mica CM05FD331J03	2101330
C12	.01μF 50V Ceramic UK-50-103	2103102
C13	.01μF 50V Ceramic UK-50-103	2103102
C14	330pF Mica CM05FD331J03	2101330
C15	560pF Mica CM05FD561J03	2101560
C16	56pF Mica CM05ED560J03	2100560
C17	56pF Mica CM05ED560J03	2100560
C18	.01μF 50V Ceramic UK-50-103	2103102
C19	.01μF 50V Ceramic UK-50-103	2103102
C20	.01μF 50V Ceramic UK-50-103	2103102
C21	.01μF 50V Ceramic UK-50-103	2103102
C22	33μF 20V Tant T362C336K020AS	2106330
C23	.01μF 50V Ceramic UK-50-103	2103102
C24	.01μF 50V Ceramic UK-50-103	2103102
C25	.01μF 50V Ceramic UK-50-103	2103102
C26	01μF 50V Ceramic UK-50-103	2103102
C27	120pF Mica CM05FD121J03	2101120
C28	4700pF 100V Ceramic CK05BX472K	2102470
C29	100pF Mica CM05FD101J03	2101100
C30	.01μF 50V Ceramic UK-50-103	2103102
C31	1μF 50V Erie 8131-050651105M	2105104
C32	.01μF 50V Ceramic UK-50-103	2103102
C33	15pF Mica CM05CD150J03	2100150
C34	.001μF Mica CM05FD102J03	2102100
C35	.002μF Mica CM06FD202J03	2102200
C36	47pF Mica CM05ED470J03	2100470
C37	1μF 35V Tant T368A105K035AS	2105102
C38	10pF Mica CM04CD100D03	2100100
C39	.01μF 50V Ceramic UK-50-103	2103102
C40	3-10pF Trim Cap Erie 538-011C-3-10pF	2510300
C41	.01μF 50V Ceramic UK-50-103	2103102
C42	33μF 20V Tant T362C336K020AS	2106330
C43	470pF Mica CM05FD471J03	2101470
C44	220pF Mica CM05FD221J03	2101220
C45	680pF Mica CM05FD681J03	2101680
C46	2.2μF 20V Tant T368A225K020AS	2105220
C47	.01μF 50V Ceramic UK-50-103	2103102
C48	.001μF Mica CM05FD102J03	2102100
C49	33pF Mica CM05ED330J03	2100330
C50	.02μF Mica CM07FD203J03	2103200
C51	1μF 35V Tant T368A105K035AS	2105102
C52	.1μF 100V Ceramic CK06BX104K	2104101
C53	100pF Mica CM05FD101J03	2101100
C54	10pF Mica CM04CD100D03	2100100

PSG-310 Parts List

Symbol	Description	Lenco Part#
C55	.01μF 50V Ceramic UK-50-103	2103102
C56	10pF Mica CM04CD100D03	2100100
C57	10pF Mica CM04CD100D03	2100100
C58	10pF Mica CM04CD100D03	2100100
C59	10pF Mica CM04CD100D03	2100100
C60	10pF Mica CM04CD100D03	2100100
C61	220μF Tant 10V Tant T362D227K010AS	2107222
C62	1μF 35V Tant T368A105K035AS	2105102
C63	1μF 35V Tant T368A105K035AS	2105102
C64	1μF 35V Tant T368A105K035AS	2105102
C65	1μF 35V Tant T368A105K035AS	2105102
C66	1μF 35V Tant T368A105K035AS	2105102
C67	1μF 35V Tant T368A105K035AS	2105102
D1	1N914 100V 10mA Signal	3100914
D2	1N914 100V 10mA Signal	3100914
D3	1N914 100V 10mA Signal	3100914
D4	Deleted	
D5	1N914 100V 10mA Signal	3100914
D6	1N914 100V 10mA Signal	3100914
D7	MV830 Varactor	3100830
D8	1N754 6.8V Zener	3100754
D9	1N754 6.8V Zener	3100754
D10	1N914 100V 10mA Signal	3100914
D11	1N757 9.1V Zener	3100757
D12	MV830 Varactor	3100830
D13	1N754 6.8V Zener	3100754
D14	1N754 6.8V Zener	3100754
D15	1N914 100V 10mA Signal	3100914
D16	1N914 100V 10mA Signal	3100914
D17	1N914 100V 10mA Signal	3100914
D18	1N914 100V 10mA Signal	3100914
D19	1N914 100V 10mA Signal	3100914
D20	1N914 100V 10mA Signal	3100914
D21	1N914 100V 10mA Signal	3100914
D22	1N914 100V 10mA Signal	3100914
D23	1N914 100V 10mA Signal	3100914
D24	1N914 100V 10mA Signal	3100914
D25	1N914 100V 10mA Signal	3100914
D26	1N914 100V 10mA Signal	3100914
D27	1N914 100V 10mA Signal	3100914
D28	1N914 100V 10mA Signal	3100914
D29	1N914 100V 10mA Signal	3100914
D30	1N914 100V 10mA Signal	3100914
D31	1N914 100V 10mA Signal	3100914
D32	1N914 100V 10mA Signal	3100914
D33	1N914 100V 10mA Signal	3100914
D34	1N914 100V 10mA Signal	3100914
D35	1N914 100V 10mA Signal	3100914
D36	1N914 100V 10mA Signal	3100914
D37	1N747 3.6V Zener	3100747
L1	82μH Delevan 1641-823	4003820
L2	22μH Delevan 2307-223	4003220
L3	10μH Miller 9310-36	4003100
L4	22μH Nytronics VIV-22	4053220
Q1	2N5457 FET-N Channel	3205457
Q2	2N3906 PNP Signal	3203906
Q3	2N3904 NPN Signal	3203904
Q4	2N3906 PNP Signal	3203906
Q5	2N3906 PNP Signal	3203906
Q6	2N3904 NPN Signal	3203904
Q7	2N3906 PNP Signal	3203906
Q8	2N5457 FET-N Channel	3205457
Q9	2N5457 FET-N Channel	3205457
Q10	2N3904 NPN Signal	3203904
Q11	2N3904 NPN Signal	3203904
Q12	2N3904 NPN Signal	3203904
Q13	2N3906 PNP Signal	3203906
Q14	2N5457 FET-N Channel	3205457
Q15	2N3904 NPN Signal	3203904
Q16	2N3904 NPN Signal	3203904
Q17	2N5457 FET-N Channel	3205457
Q18	2N3906 PNP Signal	3203906
Q19	2N3904 NPN Signal	3203904
Q20	2N5457 FET-N Channel	3205457
Q21	2N3904 NPN Signal	3203904
Q22	2N3904 NPN Signal	3203904
Q23	2N3906 PNP Signal	3203906
Q24	2N3904 NPN Signal	3203904
Q25	2N3906 PNP Signal	3203906
Q26	2N3904 NPN Signal	3203904
Q27	2N3906 PNP Signal	3203906
Q28	2N3904 NPN Signal	3203904
Q29	2N3906 PNP Signal	3203906

PSG-310 Parts List

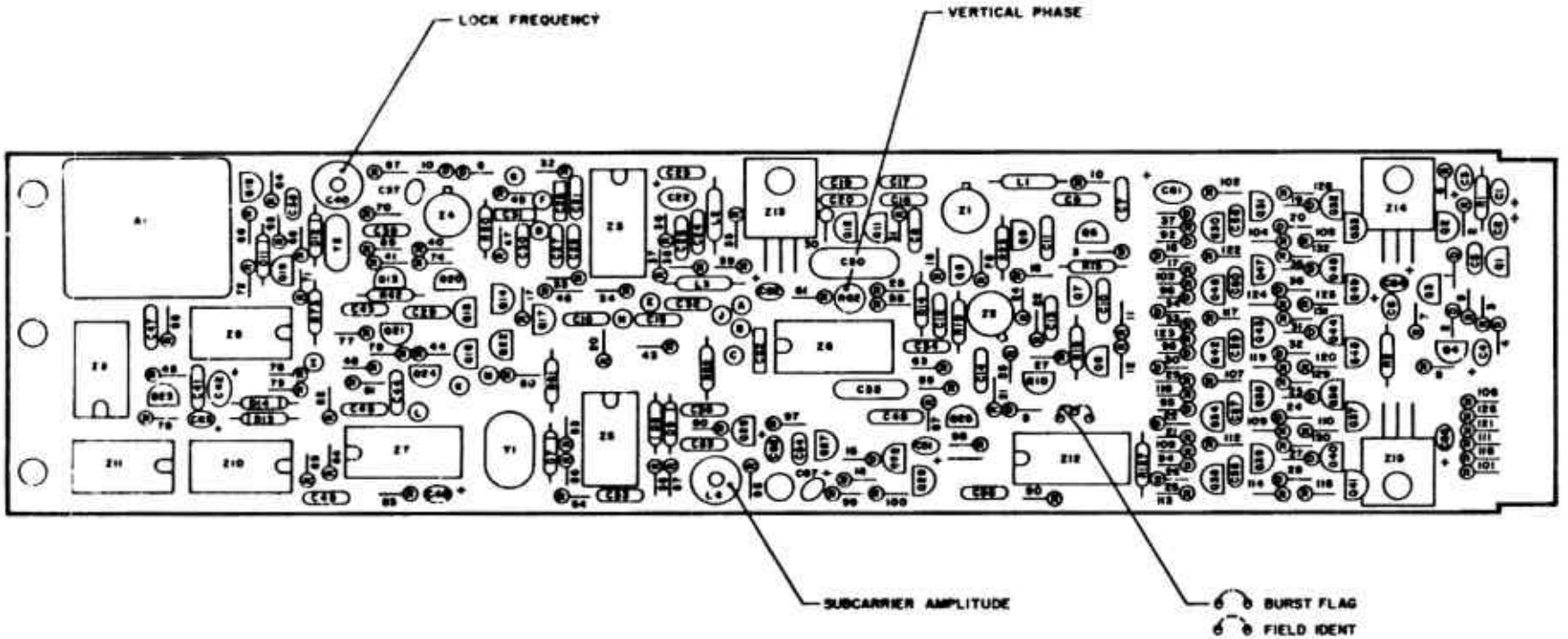
Symbol	Description	Lenco Part#
Q30	2N3904 NPN Signal	3203904
Q31	2N3906 PNP Signal	3203906
Q32	2N3904 NPN Signal	3203904
Q33	2N3906 PNP Signal	3203906
Q34	2N3904 NPN Signal	3203904
Q35	2N3906 PNP Signal	3203906
Q36	2N3904 NPN Signal	3203904
Q37	2N3906 PNP Signal	3203906
Q38	2N3904 NPN Signal	3203904
Q39	2N3906 PNP Signal	3203906
Q40	2N3904 NPN Signal	3203904
Q41	2N3906 PNP Signal	3203906
Q42	2N3904 NPN Signal	3203904
Q43	2N3906 PNP Signal	3203906
Q44	2N3904 NPN Signal	3203904
Q45	2N3906 PNP Signal	3203906
Q46	2N3904 NPN Signal	3203904
Q47	2N3906 PNP Signal	3203906
Q48	2N3904 NPN Signal	3203904
Q49	2N3906 PNP Signal	3203906
R1	1 Meg ¼w 5%	1525100
R2	510 Ohm ¼w 5%	1521510
R3	1K ¼w 5%	1522100
R4	1K ¼w 5%	1522100
R5	10K ¼w 5%	1523100
R6	27 Ohm ¼w 5%	1520270
R7	10 Ohm ¼w 5%	1520100
R8	27 Ohm ¼w 5%	1520270
R9	51 Ohm ¼w 5%	1520510
R10	390 Ohm ¼w 5%	1521390
R11	22K ¼w 5%	1523220
R12	4.7K ¼w 5%	1522470
R13	10 Meg ¼w 5%	1526100
R14	10K ¼w 5%	1523100
R15	10K ¼w 5%	1523100
R16	1 Meg ¼w 5%	1525100
R17	100K ¼w 5%	1524100
R18	100K ¼w 5%	1524100
R19	100K ¼w 5%	1524100
R20	51K ¼w 5%	1523510
R21	33K ¼w 5%	1523330
R22	2.7K ¼w 5%	1522270
R23	1K ¼w 5%	1522100
R24	2K ¼w 5%	1522200
R25	2.7K ¼w 5%	1522270
R26	1K ¼w 5%	1522100
R27	22K ¼w 5%	1523200
R28	2.2K ¼w 5%	1522220
R29	Front Panel	
R30	10K ¼w 5%	1523100
R31	10K ¼w 5%	1523100
R32	8.2K ¼w 5%	1522820
R33	15 Ohm ¼w 5%	1520150
R34	15 Ohm ¼w 5%	1520150
R35	390 Ohm ¼w 5%	1521390
R36	75 Ohm ⅛w 1%	1110750
R37	100 Ohm ¼w 5%	1521100
R38	51 Ohm ¼w 5%	1520510
R39	4.7K ¼w 5%	1522470
R40	4.7K ¼w 5%	1522470
R41	1K ¼w 5%	1522100
R42	1K ¼w 5%	1522100
R43	1K ¼w 5%	1522100
R44	1K ¼w 5%	1522100
R45	510 Ohm ¼w 5%	1521510
R46	22 Meg ¼w 5%	1526220
R47	18K ¼w 5%	1523180
R48	510 Ohm ¼w 5%	1521510
R49	51K ¼w 5%	1523510
R50	10K ¼w 5%	1523100
R51	10K ¼w 5%	1523100
R52	4.7K ¼w 5%	1522470
R53	470 Ohm ¼w 5%	1521470
R54	470 Ohm ¼w 5%	1521470
R55	Front Panel	
R56	270 Ohm ¼w 5%	1521270
R57	270 Ohm ¼w 5%	1521270
R58	1K ¼w 5%	1522100
R59	68K ¼w 5%	1523680
R60	10K ¼w 5%	1523100
R61	3.9K ¼w 5%	1522390
R62	50K Pot Beckman 82PR50K	1703500
R63	390 Ohm ¼w 5%	1521390
R64	1K ¼w 5%	1522100
R65	4.7K ¼w 5%	1522470
R66	510 Ohm ¼w 5%	1521510
R67	100K ¼w 5%	1524100
R68	1.5K ¼w 5%	1522150
R69	100K ¼w 5%	1524100
R70	100K ¼w 5%	1524100
R71	470 Ohm ¼w 5%	1521470

PSG-310 Parts List

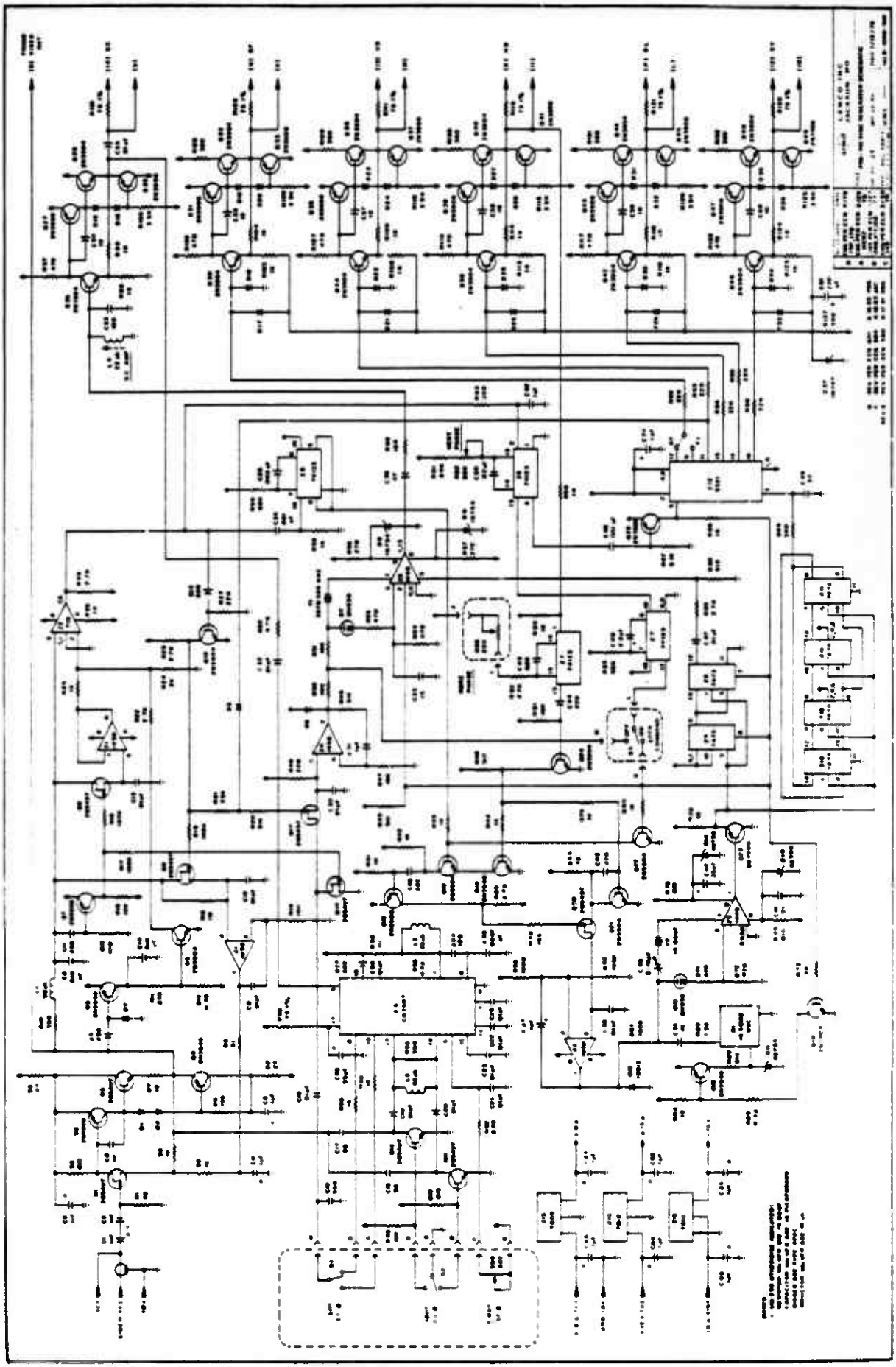
Symbol	Description	Lenco Part#
R72	470 Ohm ¼w 5%	1521470
R73	1K ¼w 5%	1522100
R74	18K ¼w 5%	1523180
R75	510 Ohm ¼w 5%	1521510
R76	510 Ohm ¼w 5%	1521510
R77	1K ¼w 5%	1522100
R78	1K ¼w 5%	1522100
R79	1K ¼w 5%	1522100
R80	1K ¼w 5%	1522100
R81	10K ¼w 5%	1523100
R82	2.7K ¼w 5%	1522270
R83	68K ¼w 5%	1523680
R84	1K ¼w 5%	1522100
R85	2.7K ¼w 5%	1522270
R86	510 Ohm ¼w 5%	1521510
R87	5.1K ¼w 5%	1522510
R88	1K ¼w 5%	1522100
R89	560 Ohm ¼w 5%	1521560
R90	1K ¼w 5%	1522100
R92	22K ¼w 5%	1523220
R93	22K ¼w 5%	1523220
R94	22K ¼w 5%	1523220
R95	22K ¼w 5%	1523220
R96	22K ¼w 5%	1523220
R97	470 Ohm ¼w 5%	1521470
R98	1K ¼w 5%	1522100
R99	1K ¼w 5%	1522100
R100	3.9K ¼w 5%	1522390
R101	75 Ohm ⅛w 1%	1110750
R102	470 Ohm ¼w 5%	1521470
R103	1K ¼w 5%	1522100
R104	1K ¼w 5%	1522100
R105	3.9K ¼w 5%	1522390
R106	75 Ohm ⅛w 1%	1110750
R107	470 Ohm ¼w 5%	1521470
R108	1K ¼w 5%	1522100
R109	1K ¼w 5%	1522100
R110	3.9K ¼w 5%	1522390
R111	75 Ohm ⅛w 1%	1110750
R112	470 Ohm ¼w 5%	1521470
R113	1K ¼w 5%	1522100
R114	1K ¼w 5%	1522100
R115	3.9K ¼w 5%	1522390
R116	75 Ohm ⅛w 1%	1110750
R117	470 Ohm ¼w 5%	1521470
R118	1K ¼w 5%	1522100
R119	1K ¼w 5%	1522100
R120	3.9K ¼w 5%	1522390
R121	75 Ohm ⅛w 1%	1110750
R122	470 Ohm ¼w 5%	1521470
R123	1K ¼w 5%	1522100
R124	1K ¼w 5%	1522100
R125	3.9K ¼w 5%	1522390
R126	75 Ohm ⅛w 1%	1110750
R127	390 Ohm ¼w 5%	1521390
R128	360 Ohm ¼w 5%	1521360
R129	360 Ohm ¼w 5%	1521360
R130	360 Ohm ¼w 5%	1521360
R131	360 Ohm ¼w 5%	1521360
R132	360 Ohm ¼w 5%	1521360
U1	MC1458CG Dual Op Amp	3301458
U2	LM710CH National Hi-Speed Comparator	3300710
U3	LM3067N Chroma Demodulator	3303067
U4	MC1458CG Dual Op Amp	3301458
U5	MC1445L 2 Ch. Switch	3301445
U6	DM74123N National Dual One-Shot	3304123
U7	DM74123N National Dual One-Shot	3304123
U8	MC1445L 2 Ch. Switch	3301445
U9	DM7473N Dual J-K Flip-Flop	3307473
U10	DM7473N Dual J-K Flip-Flop	3307473
U11	DM7473N Dual J-K Flip-Flop	3307473
U12	MM5321N National Sync Generator Chip	3305321
U13	LM7805 +5V Regulator National LM340T5	3317805
U14	LM7812 +12V Regulator National LM340T12	3317812
U15	LM7912 −12V Regulator National LM320T12	3317912
Y1	3.579545 MHz RCA 105330	4700060
Y2	14.31818 MHz HA-5	4700020
	Miscellaneous	
	I.C. Socket 16 Dip Cinch	4106120
	Printed Circuit Board (No Parts)	5041060
	Heatsink 16 Pin Series #5802B 5802B	5500100
	PSG-310 Crystal Ground Contact	5511456

FRONT PANEL
Parts List

Symbol	Description	Lenco Part#
	Front Panel	5511406
R29	20K Pot Beckman 78LBWR20K	1613200
R55	20K Pot Beckman 78LBWR20K	1613200
S1	MST-105D ALCO	4600090
S2	MST-105D ALCO	4600090
S3	MST-105D ALCO	4600090



PSG-310 SYNC GENERATOR PARTS ASSY
D-1096-110 REV B



PSG-310/PSG-311 OPTION 1

VARIABLE BLANKING WIDTH MODULE

INSTALLATION

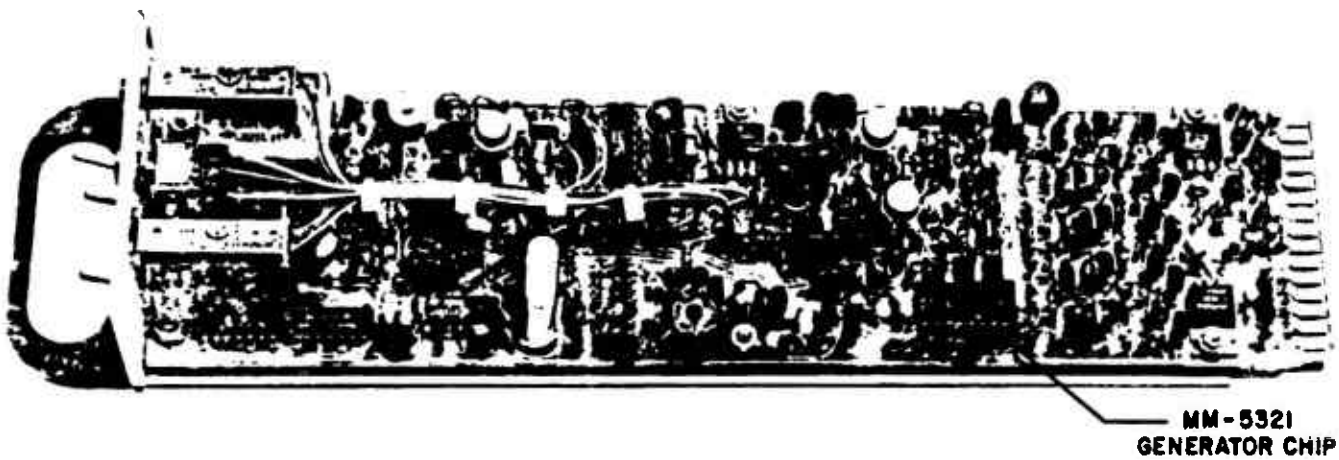
The function of the Option 1 module is to substitute e new blenking signal in plece of the one generated in the Netionel MM-5321 Sync Generetor chip on board the PSG-310 or PSG-311. It is Importent to note thet the Option 1 module preserves the leading edge timing as set up by the generator chip, but it does ellow the trailing edges to be adjusted. Another very important feeture is thet the vertical blanking widths ere chenged in half-line steps so thet the width does not very during the first ective picture line.

To instell the Option 1 module:

1. Remove the MM-5321 Sync Generator chip (Z12) from the IC socket on the PSG-310 or PSG-311 Sync Generator.
2. With a pair of needle nose pliars, carefully bend pin 14 of the MM-5321 chip outwerd end then plece the chip in the socket on the Option 1 module. When plecing the sync generetor chip in the Option 1 module, meke sure that the chip is properly oriented in the socket end not turned eround by 180°.
3. Plug the Option 1 module into the IC socket on the PSG-310 or PSG-311 Sync Generetor. Care must be taken when instelling the module to elign ell of the pins on the Option 1 module with the IC socket es it is a very tight fit.
4. **R1 - Horizontal Blanking Width** - Adjustable from approximetely 8μS to 12μS with a nominel value of 10.76μS. Set for house requirements.
5. **R12 - Vertical Blanking Width** - Adjustable from approximetely 15 lines to 22 lines with e nominal value of 20 or 21 lines. Set for house requirements.

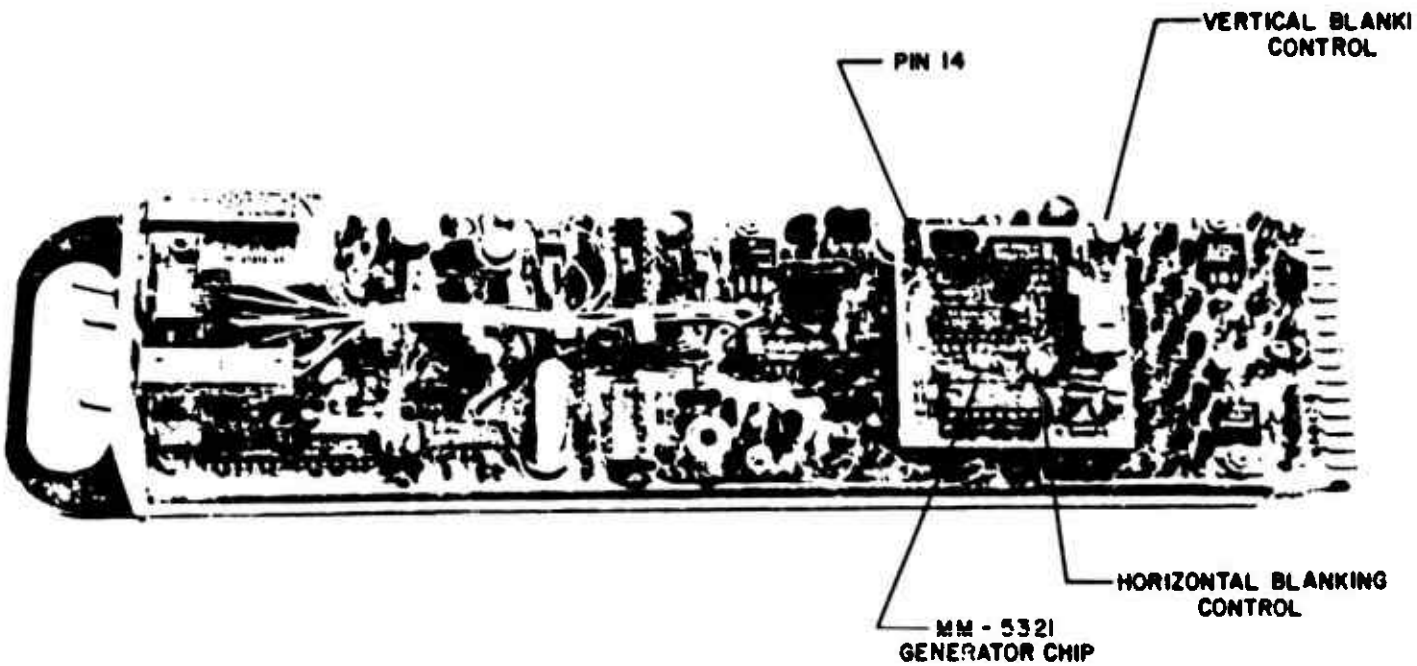
BLANKING WIDTH CONTROL ASSEMBLY

PSG-310 OR PSG-311 SYNC GENERATOR



OPTION 1
BLANKING WIDTH CONTROL ASSEMBLY

PSG-310 OR PSG-311 E/W OPTION 1



OPTION 1 - BLANKING MODULE

Adjustable Horizontal and Vertical Blanking widths are accomplished by adding the Option 1 module to the PSG-310 or PSG-311 Sync Generator. The Blanking signal from the sync generator chip, U3, pin 14, is disabled by bending the pin outward and not allowing U3 blanking to exit on pin 14 of the IC socket. Horizontal and Vertical Blanking are made out of the Horizontal Drive and Vertical Drive pulses and OR'ed together to form the composite blanking signal. The technical description will be divided into three sections for clarity.

HORIZONTAL BLANKING GENERATION

Negative going Horizontal Drive leaves U3 on pin 15 and is buffered by Q1. Q1 presents the negative going pulse to pin 11 of U1A for Horizontal Blanking generation and to pin 5 of U1B for use in Vertical Blanking generation. U1A is a CMOS one shot and it produces a negative going Horizontal Blanking pulse on pin 9 of U1A. R1, the Horizontal Blanking Width control, sets the width of the pulse on pin 9 of U1A.

VERTICAL BLANKING GENERATION

Negative going Vertical Drive leaves U3 on pin 11 and is AC coupled by C5 and presented to the base of Q2 by way of R7. Q2 inverts the pulse and Q3 reinverts the pulse again and presents the negative going pulse to pin 4 of U2A and to D4. The function of Q2 and Q3 is DC level shifting of the vertical drive pulse. U2A is a CMOS one shot and it produces a negative going Vertical Blanking pulse on pin 7 of U2A. R12, the Vertical Blanking Width control, sets the width of the pulse on pin 7 of U2A. D4 and D5 form an OR gate and mix the vertical drive pulse on the cathode of D4 with the vertical blanking pulse on the cathode of D5. The purpose of D4 is to provide the leading edge of the Vertical Blanking pulse (because of the propagation delay thru U2A) and D5 provides the trailing edge of Vertical Blanking. The Vertical Blanking pulse from the anodes of D4 and D5 are presented to the clear input, pin 13, of U2B.

The function of U1B and U2B is to make sure that when adjusting R12, the Vertical Blanking Width control, Vertical Blanking width moves in one half line steps. U1B is the second half of Horizontal Drive one shot. The time constants of R3 and C2 are set so that the output on pin 6 is slightly longer than one half of a line. This pulse is split into two signal paths and is differentiated by C4-R5 and C3-R6. This produces pulses at the leading and trailing edges of the differentiated pulse, and these signals are presented to pins 11 and 12 of U2B. U2B is a one shot with a long time constant (R14, C7) and is continuously turned on except when turned off by the Vertical Blanking pulse on pin 13. The output of U2B, pin 10, is a negative going Vertical Blanking pulse that can only change in width at half line steps.

COMPOSITE BLANKING GENERATION

The Composite Blanking pulse is generated by OR'ing the Horizontal Blanking pulse and Vertical Blanking pulse together. D1 and D2 provide the Horizontal Blanking pulse with D1 making up the leading edge of the pulse (because of the propagation delay thru U1A) and D2 provides the trailing edge of the Horizontal Blanking pulse. D6 provides the Vertical Blanking pulse and the Composite Blanking pulse leaves the socket of U3 on pin 14.

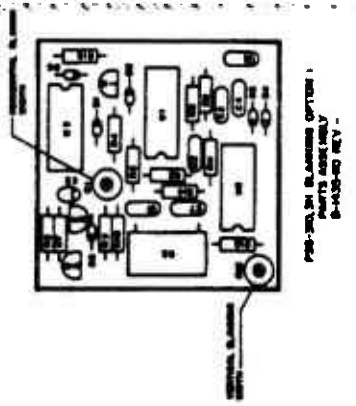
PSG-310/PSG-311 OPTION 1

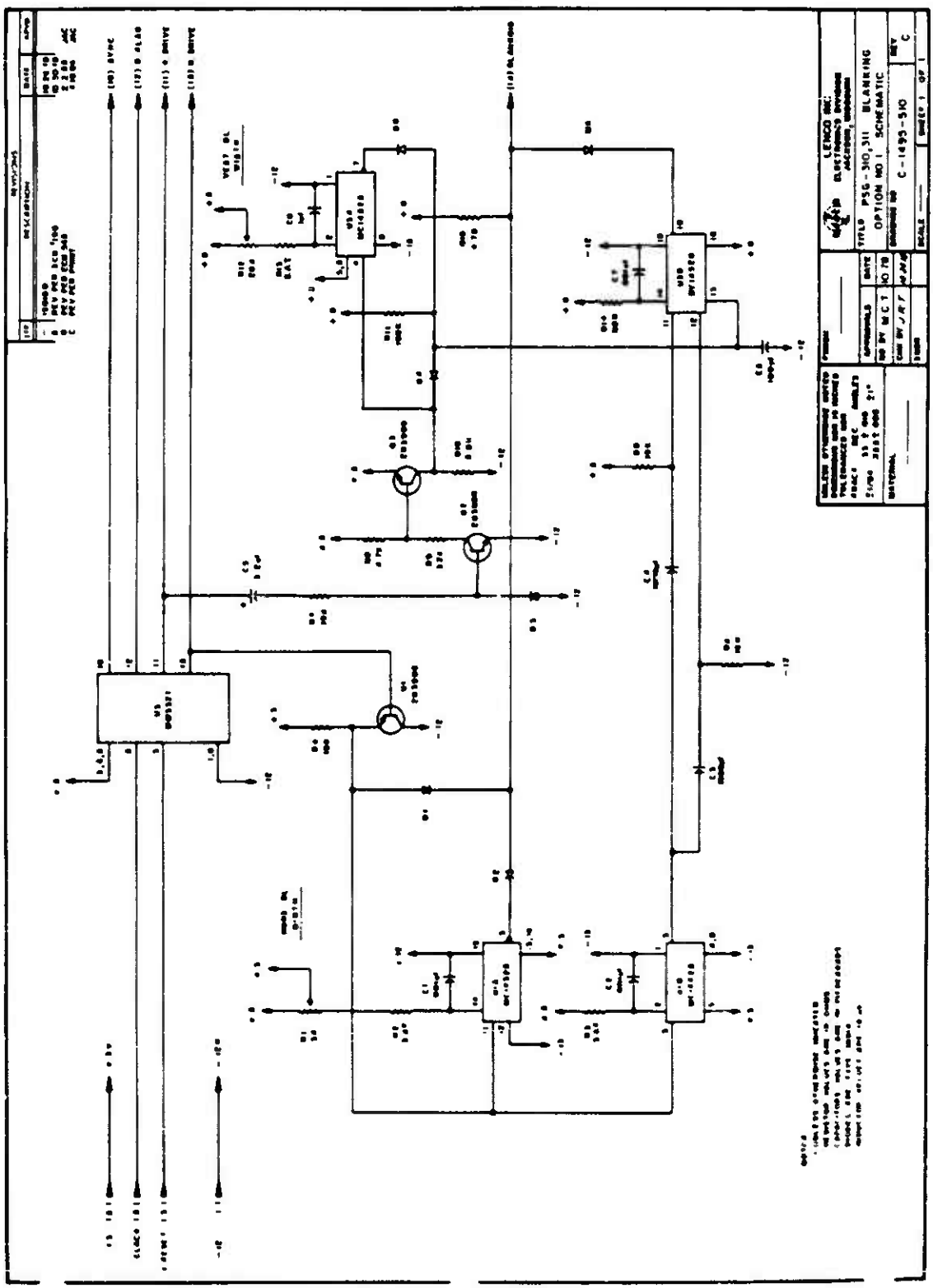
VARIABLE BLANKING WIDTH MODULE

Parts List

Symbol	Description	Lenco Part#
C1	.001μF Mica CM05FD102J03	2102100
C2	.001μF Mica CM05FD102J03	2102100
C3	.001μF Mica CM05FD102J03	2102100
C4	.001μF Mica CM05FD102J03	2102100
C5	2.2μF 20V Tant T368A225K020AS	2105220
C6	.1μF 100V Film 225P10491WD3	2104103
C7	.001μF Mica CM05FD102J03	2102100
C8	100pF Mica CM05FD101J03	2101100
D1	1N914 100V 10mA Signal	3100914
D2	1N914 100V 10mA Signal	3100914
D3	1N914 100V 10mA Signal	3100914
D4	1N914 100V 10mA Signal	3100914
D5	1N914 100V 10mA Signal	3100914
D6	1N914 100V 10mA Signal	3100914
Q1	2N3906 PNP Signal	3203906
Q2	2N3904 NPN Signal	3203904
Q3	2N3906 PNP Signal	3203906
R1	5K Pot Spectrol 62-1-1-502	1702500
R2	13K ¼w 5% S.A.T.	1523130
R3	56K ¼w 5% S.A.T.	1523560
R4	10K ¼w 5%	1523100
R5	10K ¼w 5%	1523100
R6	10K ¼w 5%	1523100
R7	10K ¼w 5%	1523100
R8	4.7K ¼w 5%	1522470
R9	22K ¼w 5%	1523220
R10	3.9K ¼w 5%	1522390
R11	100K ¼w 5%	1524100
R12	20K Pot Spectrol 62-1-1-203	1703200
R13	15K ¼w 5% S.A.T.	1523150
R14	56K ¼w 5%	1523560
R15	4.7K ¼w 5%	1522470
U1	MC14528BCP Dual One Shot	3304528
U2	MC14528BCP Dual One Shot	3304528
	ME-2B-16-WC-B10 16 Pin Dip Socket	4106121
	Printed Circuit Board (No Parts)	5021434

S.A.T. = Selected At Test





PRODUCT WARRANTY

Lenco, Inc., Electronics Division warrants each new product manufactured by it, to be free from defective material and workmanship and agrees to remedy any such defect either by repair or replacement at no charge for a period of two years from the date of original shipment.

This Warranty does not extend to any Lenco products which have been subjected to misuse, neglect, accident, incorrect wiring, improper installation, or used in violation of instructions by Lenco, nor extend to equipment which have been altered outside Lenco's factory, without prior approval in writing, nor to equipment where the serial number has been removed, defaced or changed, nor to accessories used therewith, not manufactured by Lenco.

Cathode Ray Tubes (CRT) used in products manufactured by Lenco are warranted for a period of one (1) year from the date of original purchase. CRT's covered by the Warranty must be delivered with proper packaging to prevent damage. This Warranty does not extend to any CRT that is received at the Lenco factory that has broken glass.

Equipment covered by the Warranty must be delivered by the owner, with all transportation charges prepaid, to Lenco's factory for examination. If examination discloses, in Lenco's judgement, that it is thus defective, the equipment will be repaired or replaced. Equipment returned prepaid under Warranty and repaired in Lenco's plant will be returned with all transportation charges, surface freight only, paid by Lenco. Units that fail under conditions other than those covered above, will be repaired on a cost of components, plus labor basis. All freight charges of equipment to be repaired, not under Warranty, will be the responsibility of the owner.

This Warranty is in lieu of any other Warranty, obligation, or liability, expressed or implied, including any Warranty of merchantability or fitness for a particular purpose, with respect to any and all equipment furnished by Lenco. No representative or person is authorized to assume for Lenco any other liability in connection with the sale of its products.

Under no circumstances shall Lenco, Inc., be liable, in contracts or in tort, for any economic loss, including any loss of profits, or for any special or consequential damage or loss arising from any cause.

All inquiries relating to either parts replacement or Warranty service shall be directed to: Lenco, Inc., Electronics Division, 300 N. Maryland St., Jackson, Missouri, 63755, attention: Customer Service — Phone 314-243-3147.

APPENDIX E

WESTERN TELEMATICS, INC. CAS-41 MANUAL

OPERATING AND INSTALLATION INSTRUCTIONS
MODEL CAS-41
RS232 CODE ACTIVATED TERMINAL SWITCH

INTRODUCTION

The model CAS-41 CODE ACTIVATED RS232 asychronous Terminal Switching Device is a self-powered unit which, when connected to a standard modem or CPU, can switch between any combination of RS232 terminal ports. Terminal status and modem operation can be tested by using the ACK/NAK answerback and Data Loopback features.

OPERATION

The CAS-41 is programmed by a 2 ASCII character code sequence received from the Modem port. The first character is a user selectable prefix code which is programmed by setting switch S3 on the circuit board for the desired ASCII code. This character when received resets the unit and closes all terminal ports. The next character received configures the CAS-41 for the desired operation (Refer to Table 3) and allows data to flow in both directions until the next prefix code is detected.

CONNECTION

Before connecting, Refer to Table 1 and check the interface requirements of the CAS-41, Terminals and Modem to assure proper dataflow and avoid conflicts of inputs or outputs being on the same pin.

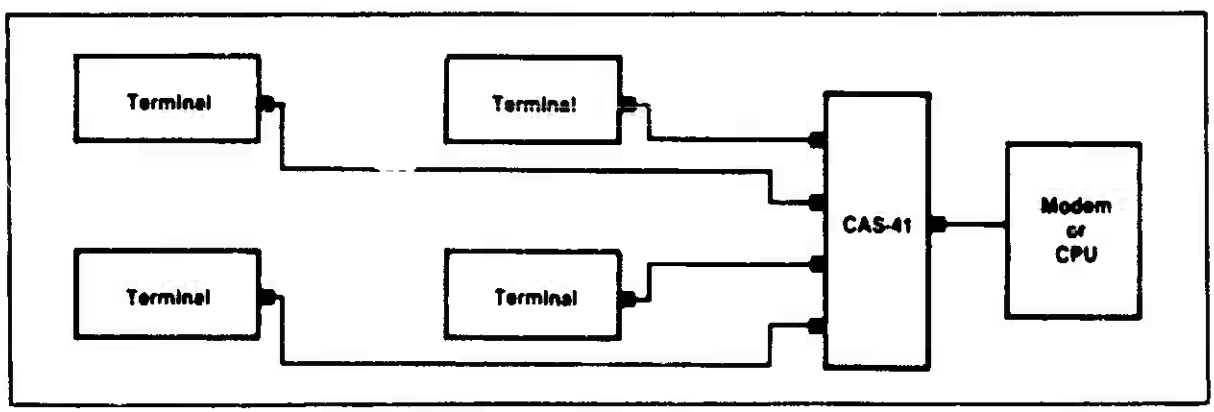


TABLE 1 INTERFACE REQUIREMENTS

PIN	TERMINAL SIGNAL	PORTS 1/0	NOTE	PIN	MODEM SIGNAL	PORTS 1/0	NOTE
1	Ground	---		1	Ground	---	
2	TX Data	In		2	TX Data	Out	
3	RX Data	Out		3	RX Data	In	
4	Req. to Send	In	2	4	Req. to Send	Out	
5	Clear to Send	Out	3	5	Clear to Send	In	
6	Data Set Rdy.	Out	1	7	Ground	---	
7	Ground	---		8	Carrier Detect	In	
8	Carrier Detect	Out	4	20	Terminal Ready	Out	1
20	Terminal Ready	In	2				

Notes:

1. Always High (+) supplied by CAS-41
2. Select pin 4 or 20 for ACK/NAK feature on J1 thru J4 on PC Board
3. Force High (+) by switching S2-7 ON
4. Force High (+) by switching S2-8 ON

SWITCH SET-UP

To access the configuration switches, remove the four black screws attaching the plastic cover to the chassis and remove the cover.

Select the baud rate and parity settings as shown in Table 2 and 3. Select the PREFIX code, ACK/NAK, Loopback and Interface options as shown in Table 3.

TABLE 3 BAUD RATE SELECT - SWITCH S1

Select only one ON, all others OFF

Position	Baud Rate	
1	19200	
2	9600	
3	4800	
4	2400	
5	1200	For 110 baud operation
6	600	also switch S2-1 OFF
7	300	
8	75/110	

TABLE 2 SWITCH 2 SET-UP

Position	On	Off
1	75-19.2K BAUD	110 BAUD
2	1 STOP BIT	2 STOP BITS
3	PARITY	
4	PARITY	
5	INHIBIT ACK/NAK	NORMAL
6	INHIBIT LOOPBACK	NORMAL
7	FORCE CTS	NORMAL
8	FORCE DCD	NORMAL

3	4	PARITY
ON	ON	ODD
ON	OFF	EVEN
OFF	ON	SPACE
OFF	OFF	MARK

PREFIX CODE SELECTION

Internal Switch S3 selects the desired Prefix code. Switch 1 thru 7 corresponds to bit 1 thru 7 of an ASCII code. All switches should be ON except for the bits selected. For example, to select CTRL A (SOH) as the prefix code switch 1 should be OFF and all others ON. For CTRL Y (EM), switch 1, 4 and 5 should be OFF. A non-printing character that does not affect the terminals operation (Keyboard Lock etc.) should be selected since the prefix code will be transmitted thru any port previously selected.

CONFIGURATION CODE SELECTION

The configuration code follows the prefix code and activates the desired terminal ports and selects the ACK/NAK and Loopback features. The configuration code is not transmitted to the terminal since all terminal ports are closed after receipt of the prefix code and will not be opened again until after the configuration code.

Bits 1, 2, 3 and 4 of the desired ASCII code corresponds to Terminal ports 1, 2, 3 and 4. Bit 5 selects the ACK/NAK feature and bit 6 selects the Loopback feature.

Table 4 lists all 128 ASCII codes with the corresponding action of the CAS-41. If the ACK/NAK or Loopback features are disabled by switch S2-5 and 6 these functions will not be selected. Disabling these features increases the port select characters available. For example, eight different codes could be used to select port 1.

Do not select a configuration code that is also used as the prefix code since the unit resets on each prefix code.

TABLE 4

Configuration Code Chart

Code	Terminal	ACK/NAK	Loopback	Code	Terminal	ACK/NAK	Loopback	Code	Terminal	ACK/NAK	Loopback
NULL	ALL OFF	NO	NO	+	1 2 4	NO	YES	V	2 3	YES	NO
SOH	1	NO	NO	,	3 4	NO	YES	W	1 2 3	YES	NO
STX	2	NO	NO	-	1 3 4	NO	YES	X	4	YES	NO
ETX	1 2	NO	NO	.	2 3 4	NO	YES	Y	1 4	YES	NO
EOT	3	NO	NO	/	1 2 3 4	NO	YES	Z	2 4	YES	NO
ENQ	1 3	NO	NO	0	ALL OFF	YES	YES	[	1 2 4	YES	NO
ACK	2 3	NO	NO	1	1	YES	YES	\	3 4	YES	NO
BELL	1 2 3	NO	NO	2	2	YES	YES	]	1 3 4	YES	NO
BS	4	NO	NO	3	1 2	YES	YES	^	2 3 4	YES	NO
HT	1 4	NO	NO	4	3	YES	YES	_	1 2 3 4	YES	NO
LF	2 4	NO	NO	5	1 3	YES	YES	`	ALL OFF	NO	YES
VT	1 2 4	NO	NO	6	2 3	YES	YES	a	1	NO	YES
FF	3 4	NO	NO	7	1 2 3	YES	YES	b	2	NO	YES
CR	1 3 4	NO	NO	8	4	YES	YES	c	1 2	NO	YES
SO	2 3 4	NO	NO	9	1 4	YES	YES	d	3	NO	YES
SI	1 2 3 4	NO	NO	:	2 4	YES	YES	e	1 3	NO	YES
DLE	ALL OFF	YES	NO	;	1 2 4	YES	YES	f	2 3	NO	YES
OC1	1	YES	NO	<	3 4	YES	YES	g	1 2 3	NO	YES
OC2	2	YES	NO	=	1 3 4	YES	YES	h	4	NO	YES
OC3	1 2	YES	NO	>	2 3 4	YES	YES	i	1 4	NO	YES
OC4	3	YES	NO	DEL	1 2 3 4	YES	YES	j	2 4	NO	YES
NAK	1 3	YES	NO	@	ALL OFF	NO	NO	k	1 2 4	NO	YES
SYN	2 3	YES	NO	A	1	NO	NO	l	3 4	NO	YES
ETB	1 2 3	YES	NO	B	2	NO	NO	m	1 3 4	NO	YES
CAN	4	YES	NO	C	1 2	NO	NO	n	2 3 4	NO	YES
EM	1 4	YES	NO	D	3	NO	NO	o	1 2 3 4	NO	YES
SUB	2 4	YES	NO	E	1 3	NO	NO	p	ALL OFF	YES	YES
ESC	1 2 4	YES	NO	F	2 3	NO	NO	q	1	YES	YES
FS	3 4	YES	NO	G	1 2 3	NO	NO	r	2	YES	YES
GS	1 3 4	YES	NO	H	4	NO	NO	s	1 2	YES	YES
RS	2 3 4	YES	NO	I	1 4	NO	NO	t	3	YES	YES
US	1 2 3 4	YES	NO	J	2 4	NO	NO	u	1 3	YES	YES
SP	ALL OFF	NO	YES	K	1 2 4	NO	NO	v	2 3	YES	YES
!	1	NO	YES	L	3 4	NO	NO	w	1 2 3	YES	YES
"	2	NO	YES	M	1 3 4	NO	NO	x	4	YES	YES
#	1 2	NO	YES	N	2 3 4	NO	NO	y	1 4	YES	YES
$	3	NO	YES	O	1 2 3 4	NO	NO	z	2 4	YES	YES
%	1 3	NO	YES	P	ALL OFF	YES	NO	{	1 2 4	YES	YES
&	2 3	NO	YES	Q	1	YES	NO	|	3 4	YES	YES
'	1 2 3	NO	YES	R	2	YES	NO	}	1 3 4	YES	NO
(	4	NO	YES	S	1 2	YES	NO	~	2 3 4	YES	YES
)	1 4	NO	YES	T	3	YES	NO	?	1 2 3 4	YES	YES
*	2 4	NO	YES	U	1 3	YES	NO				

ACK/NAK ANSWERBACK

The ACK/NAK feature provides a means for the computer to determine if a data terminal is connected to the selected port or a device is ready to send or receive data.

Bit 5 of the configuration code selects the ACK/NAK feature unless switch S2-5 is ON. Immediately after the configuration code is received the CAS-41 monitors pin 20 (Terminal Ready) of the selected terminal port and sends to the computer an ACK code if the signal is high (+) and an NAK code if the signal is low (-). Jumpers (J1 thru J4) are provided to monitor pin 4 (Request to Send) instead of pin 20.

LOOPBACK

The Loopback feature provides a means for the computer to perform a data test by receiving and comparing data it sent via the transmission line. Bit 6 of the configuration code selects the Loopback feature unless S2-6 is ON. When selected the CAS-41 will transmit data received from the modem port back out the modem port. All terminal ports and the ACK/NAK features should be switched OFF by using a space as the configuration code.

To program the Prefix Code, set Switch 3 for the desired ASCII code. Switch position 1 thru 7 corresponds to ASCII Bits 1 thru 7. The switch should be ON for a 0 and OFF for a 1.

b7 b6 b5 / Bits					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	ROW / COLUMN	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

STATUS INDICATORS

LED indicators on the front panel of the unit display the status of the terminal ports and communication lines.

Terminal 1 - 4	Indicates terminal port selected.
RX Data	Indicates data activity on the Received Data Line of the Modem Port, pin 3.
TX Data	Indicates data activity on the Transmit Data Line of the Modem Port, pin 2.
DCD	Indicates status of the Data Carrier Detect signal (pin 8) of the Modem Port. DCD may be forced on by S2-8 in the ON position.
CTS	Indicates status of the Clear to Send signal (pin 5) of the Modem Port. CTS may be forced ON by S2-7 in the ON position.

WARRANTY

One year repair or replacement at no charge unless unit is damaged due to mishandling or improper use. Call WTI for Return Authorization.

CUSTOMER SERVICE

Please contact our Customer Service Department for more detailed information or technical assistance.

(714) 979-0363 or
(800) 854-7225 Outside California Only

Western Telematic, Inc.
2435 S. Anne Street
Santa Ana, CA 92704